UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

TRABAJO FIN DE GRADO

Clasificación de flujos de tráfico en Internet utilizando técnicas de aprendizaje automático

Autor: Alejandro Romero del Campo
Tutor: Luis de Pedro Sánchez
Ponente: Jorge Enrique López de Vergara Méndez

Junio 2019

Clasificación de flujos de tráfico en internet utilizando técnicas de aprendizaje automático

AUTOR: Alejandro Romero del Campo
TUTOR: Luis de Pedro Sánchez

Departamento TEC
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Junio de 2019

Resumen

El objetivo principal de este trabajo de fin de grado es la clasificación de aplicaciones y flujos de tráfico en Internet mediante el uso de técnicas de aprendizaje automático e inteligencia artificial. Después, tras haber obtenido unos resultados satisfactorios, se estudiará la posibilidad de aplicar estos resultados para un uso más específico como es la clasificación y detención de malware.

Técnicas como la clasificación de aplicaciones mediante el número de puerto o la inspección de paquetes (DPI) son técnicas un tanto obsoletas ya o que plantean el problema de la privacidad. Sin embargo, durante este trabajo estudiaremos la posibilidad usar técnicas de aprendizaje automático para solucionar estos problemas.
Para ello utilizaremos la herramienta Weka [6]. Esta herramienta, como ya se observará más adelante incluye una gran cantidad de técnicas de clasificación y aprendizaje automático.

Para poder realizar una fase de entrenamiento lo suficientemente ajustada a la realidad se contará con conjuntos de datos cedidos por la Universidad Politécnica de Cataluña.
Uno de los problemas que se nos plantea es la adaptación de estos conjuntos al formato Weka para así poder realizar la clasificación correcta. Para ello se estudiará este formato y se desarrollarán herramientas para poder adaptar los conjuntos de datos.

Se realizarán diferentes pruebas variando el tipo de clasificador, el conjunto de datos (de los 7 cedidos por la UPC) y filtrado de diferentes campos hasta encontrar el conjunto de entrenamiento y test que mejores resultados proporciona para una correcta clasificación de aplicaciones a partir de datos estadísticos de flujos de red.

Por último, y tras haber obtenido resultados satisfactorios, se aplicarán estos resultados para una clasificación y detención de malware. Para ello se va a generar tráfico malware simulado y con diferentes herramientas desarrolladas se adaptará este tráfico malicioso junto con los conjuntos de datos de la UPC al formato Weka.
Una vez hecho esto y con las características de clasificación que resultaron más óptimas en la fase anterior, se generará un clasificador para una detención de malware lo más óptimo posible.

Palabras clave

Clasificación de flujos, tráfico, conjuntos de datos, inteligencia artificial, aprendizaje automático, clasificación, tráfico malicioso, numero de puerto, flujos, paquetes…

Abstract

The main objective of this Bachelor Thesis is the classification of applications and flows of traffic on the Internet through the use of automatic learning techniques and artificial intelligence. Then, after having obtained satisfactory results, we will study the possibility of applying these results for a more specific use such as the classification and detention of malware.

Techniques such as the classification of applications through the port number or the inspection of packages (DPI) are obsolete techniques already or supposes a problem of privacy. However, during this work we will study the possibility of using automatic learning techniques to solve these problems. For this we will use the Weka tool [6]. This tool, as will be seen later, includes a large number of classification and automatic learning techniques.

In order to accomplish a training phase sufficiently adjusted to reality, there will be datasets provided by the Polytechnic University of Catalonia. One of the problems that we face is the adaptation of these sets to the Weka format in order to make the correct classification. For this purpose, this format will be studied and tools will be developed to adapt the data sets.

Different tests will be carried out varying the type of classifier, the data set (of the 7 assigned by UPC) and filtering of different fields until finding the training and test set that provides the best results for a correct classification of applications based on data statistics of network flows.

Finally, and after having obtained satisfactory results, these results will be applied for a classification and detention of malware. For this purpose, simulated malware traffic will be generated and, with different tools developed, this malicious traffic will be adapted together with UPC data sets to the Weka format.
Once this is done, and with the classification characteristics that were most optimal in the previous phase, a classifier will be generated to detect malware, as optimal as possible.

Keywords

Flow classification, traffic, data sets, artificial intelligence, machine learning, classification, malware, port number, flow, packets…

Agradecimientos

Agradecer a la Universidad Politécnica de Cataluña (UPC) por la cesión de todos los conjuntos de datos que han sido utilizados para poder construir los clasificadores y que sin duda han sido de vital importancia para la realización de este trabajo de fin de grado.

Agradecer a mi tutor Luis de Pedro Sánchez y a mi ponente Jorge Enrique López de Vergara Méndez.

Agradecer a Cristina por tantas horas de biblioteca y muchos ánimos.

Y, por último, agradecer a mis padres por todo el apoyo proporcionado para que este trabajo fuera posible.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

GLOSARIO

ACK	Acknowledgement TCP flag
API	Application Programming Interface
APP	Application
ARFF	Attribute Relation File Format
DDOS	Distribute Denial of Service
DNS	Domain Name System
DPI	Deep Packet Inspection
FIN	End of connection TCP flag
FTP	File Transport Protocol
HTTP	Hyper Text Transfer Protocol
IP	Internet Protocol
PSH	Push TCP flag
P2P	Peer to Peer
RST	Reset TCP flag
SMTP	Simple Mail Transfer Protocol
SYN	Synchronization TCP flag
TCP	Transmission Control Protocol
ToS	Type of Service
UAM	Universidad Autónoma de Madrid
UPC	Universidad Politécnica de Cataluña
URG	Urgent TCP flag

1 Introducción

1.1 Motivación

Existen multitud de técnicas para la clasificación de tráfico y flujos. Todas estas técnicas poseen ventajas, como se comentará en la descripción del estado del arte, pero también grandes desventajas. La motivación de este trabajo de fin de grado es observar las ventajas que plantea el uso de técnicas de aprendizaje automático para una clasificación de flujos de tráfico en internet mucho más óptima.

Todo esto, solucionando problemas que se comentarán más adelante como lo son la privacidad de los flujos. Pero las motivaciones de este trabajo de fin de grado no quedan solo en la posibilidad de mejorar la clasificación de tráfico en internet, si no estudiar las posibles aplicaciones que podría tener.

Es por ello por lo que se ha estudiado su aplicación en la detención de tráfico malicioso y ataques. Las mejoras de la inteligencia artificial y técnicas de aprendizaje automático suponen una gran ventaja en el ámbito de la seguridad, debido a los conocimientos que se poseen sobre los diferentes tipos de ataques y como esta información puede ser de gran utilidad a la hora de detectarlos en un futuro construyendo un clasificador fiable.

Existen multitud de aplicaciones más de las que son más comunes y conocidas para la sociedad, derivadas de las técnicas de aprendizaje automático y, sin duda, aplicarlas al ámbito de la clasificación de flujos es de gran utilidad y posee grandes perspectivas de futuro.

Con la ayuda de conjuntos de datos lo suficientemente fiables, como los que han sido cedidos por la UPC, la motivación principal de este trabajo de fin de grado es el ajuste de estos conjuntos de datos, así como la programación de unas técnicas de simulación, para poder crear un clasificador, analizando y estudiando los resultados que se obtienen, que se ajuste al máximo a la realidad, con una alta fiabilidad probada y testeada, para, después, aplicar todos los resultados aprendidos para construir un detector de ataques efectivo, creado junto con los conjuntos de datos para que sea similar al tráfico que se podría capturar y encontrar en cualquier situación real en internet.

1.2 Objetivos

El objetivo principal de este trabajo de fin de grado es la creación y estudio de un clasificador de flujos en internet fiable y que realmente se ajuste a la realidad para así, poder clasificar aplicaciones y estudiar los diferentes parámetros y factores que son más relevantes para clasificar cada una de las aplicaciones individualmente.

Una vez elaborado un clasificador óptimo, se procederá a aplicar estos resultados para una aplicación en concreto, la creación de un clasificador capaz de detectar ciertos tipos de ataques con los que ha sido entrenado, en tráfico totalmente real.

Durante este trabajo se ha contado con los conjuntos de datos cedidos por la UPC y con el software libre para algoritmos de aprendizaje automático, Weka.
A partir de esto, el trabajo ha consistido en estudiar y adaptar los data sets al formato aceptado por Weka. Después se ha ajustado y parametrizado el software Weka para poder

crear el modelo de clasificación deseado y, tras ello, se han realizado varios tipos de simulaciones y filtrados de los conjuntos de datos hasta encontrar el mejor resultado posible, analizando que tipos de flujos proporcionan mejores resultados y sus motivos, en términos de precisión de clasificación.

Para crear un clasificador que detecte tráfico malicioso, teniendo ya el modelo de clasificación y filtrado más óptimo, se utilizarán capturas de tráfico malware (algunas obtenidas en internet de forma libre y otras generadas para la realización de este trabajo) para construir un nuevo clasificador que sea capaz de detectar tráfico malicioso. Se realizarán pruebas reales para comprobar la eficacia de este clasificador.

1.3 Planificación

A continuación, a modo de diagrama de *Gantt*, queda representado, en una línea temporal, todo el proceso de realización del presente trabajo de fin de grado, así como una serie de diagramas en los que se puede apreciar, de manera clara, el trabajo que se ha llevado a cabo durante la elaboración de este trabajo de fin de grado.

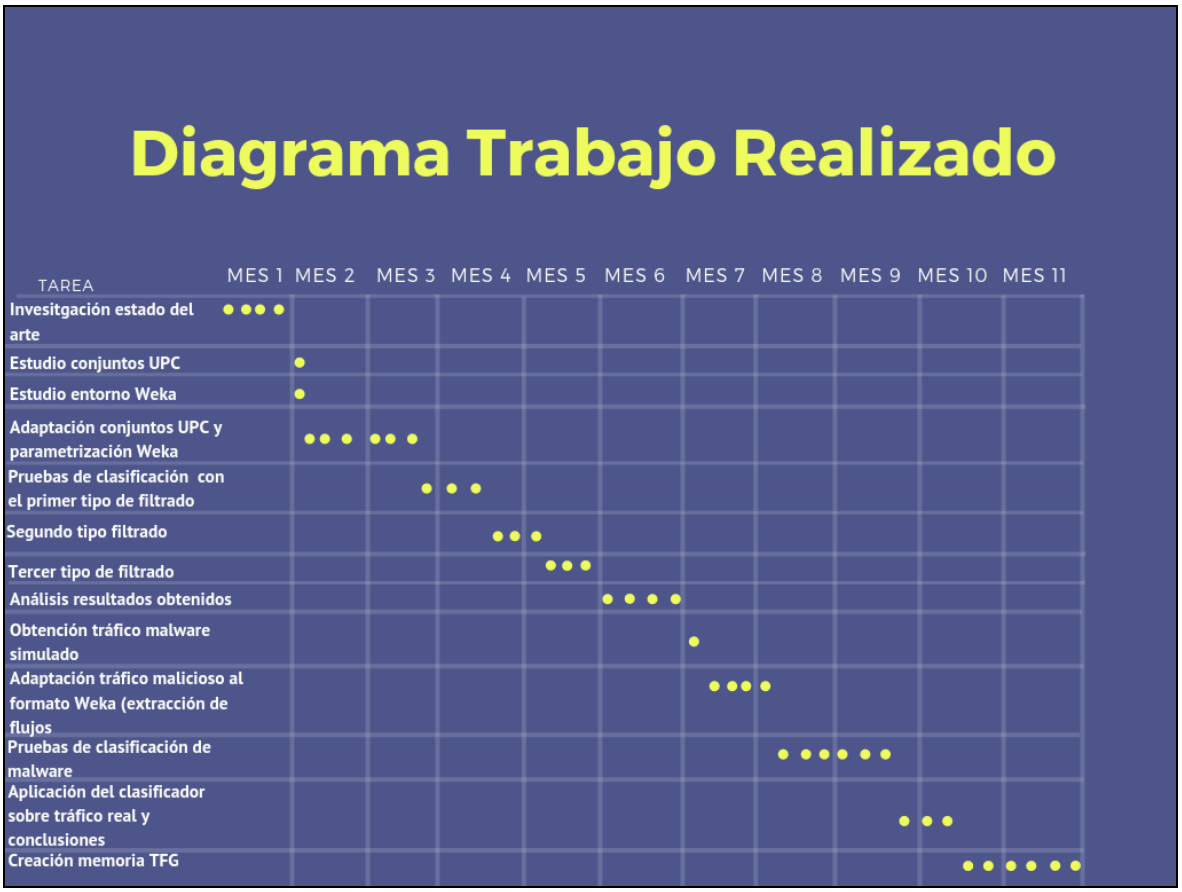

Figura 1-1: ***Diagrama Gantt trabajo realizado.***

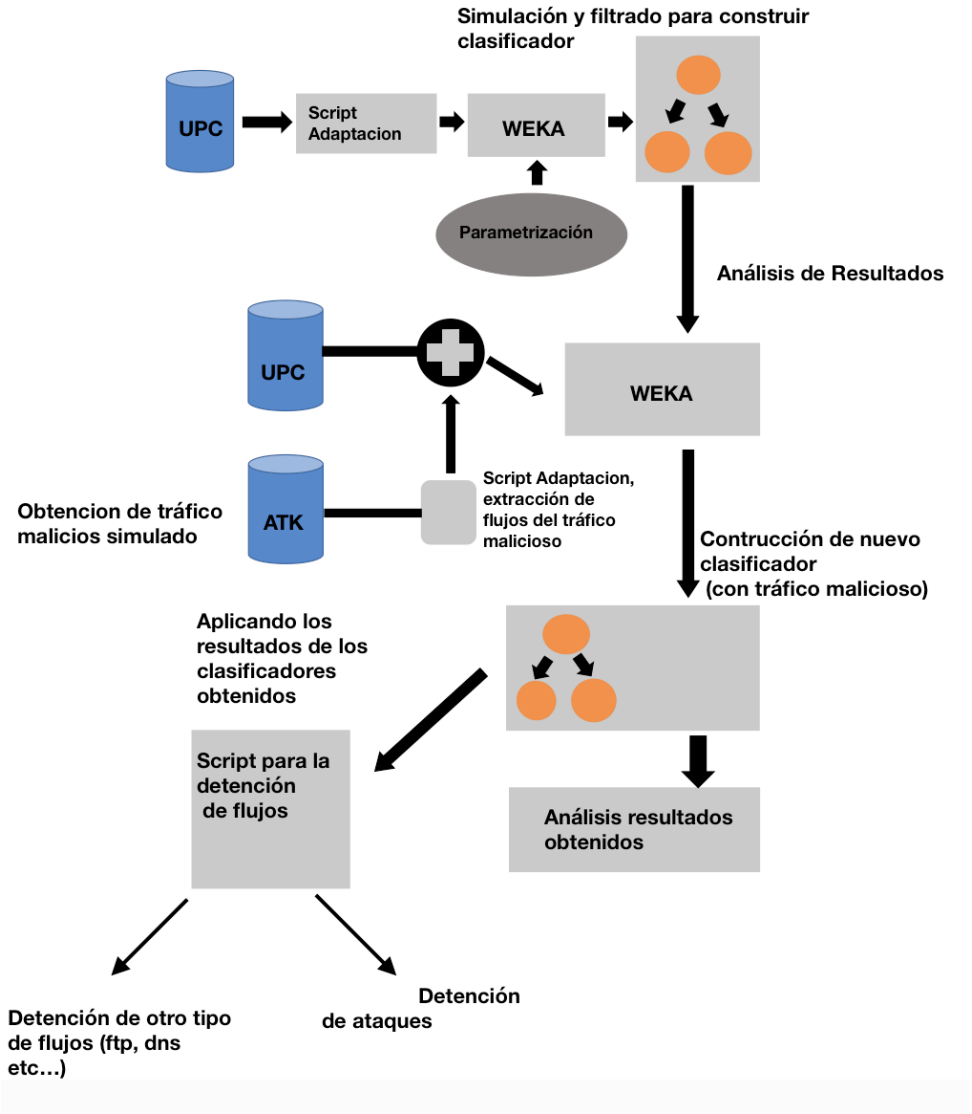

Figura 1-2: ***Diagrama proceso del trabajo realizado***

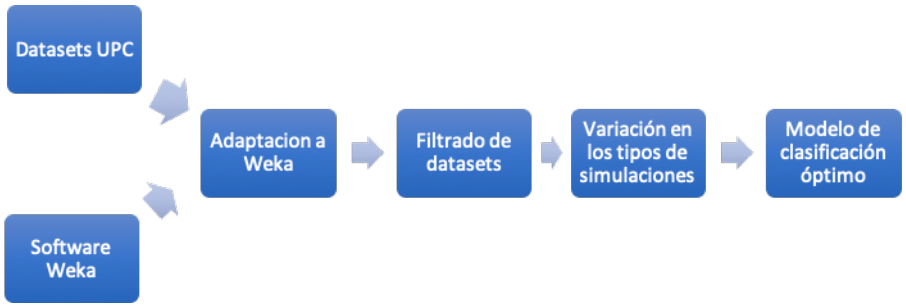

Figura 1-3: ***Proceso clasificación de flujos de tráfico***

Figura 1-4: ***Proceso aplicación concreta de los resultados de la clasificación de flujos***

1.4 Organización de la memoria

La memoria consta de los siguientes capítulos:

- Introducción: Se explican las motivaciones y objetivos detrás del presente trabajo y se describe el trabajo que se ha realizado.
- Estado del arte: Breve descripción sobre métodos de clasificación actuales y pasados, así como una comparación con las técnicas de aprendizaje automático. También se explicará la razón del uso de flujos de tráfico.
- Diseño: Durante este capítulo se realizará una breve descripción de los conjuntos de datos que se usarán, así como del entorno Weka. También se explicará la forma en la que estos conjuntos de datos deben ser adaptados para ser tratados por el software Weka.
- Desarrollo de árboles de clasificación de aplicaciones: En este capítulo se realizan tres filtrados distintos de los data sets, así como todos los tipos distintos de simulaciones hasta conseguir un clasificador con los mejores resultados. Estos resultados serán analizados y, también, para cada aplicaciones y flujo individualmente. Una vez obtenidos unos resultados satisfactorios, su utilizarán para la construcción de un clasificador capaz de detectar ataques. Para ello se realizarán diferentes ataques simulados y se realizarán herramientas para extraer los flujos de tráfico y adaptarlos al formato Weka.
- Conclusiones y trabajo futuro: Se comentarán las conclusiones y resultados del trabajo realizado, así como una perspectiva de futuro.
- Referencias: Bibliografía utilizada.
- Glosario: palabras clave explicadas.
- Anexos: Parte donde quedan explicados aspectos no tan necesarios para el entendimiento de este trabajo o que se alejan ligeramente del tema principal pero que han llegado a ser necesarios. También se incluyen los resultados que nos proporciona el programa Weka, sin ser adaptados para la memoria.

2 Estado del arte

2.1 Introducción

Durante esta sección se expondrá el estado del arte acerca de los métodos y técnicas de clasificación que son importantes para poder entender los métodos de aprendizaje automático utilizados en este trabajo.

Se van a describir cuales son los factores que debe cumplir un clasificador para que este sea lo más óptimo posible tanto en el uso de recursos, como en problemas con la privacidad de los usuarios de la red. Se hablará de técnicas como la clasificación de tráfico mediante el uso de puertos o la técnica *Deep Packet Inspection (DPI)* y de que mejoras plantea el uso de técnicas de aprendizaje automático.
Finalmente se justificará el hecho de por qué, para la realización de este trabajo y sus diferentes pruebas, se han usado flujos de tráfico y no paquetes de forma individual.

2.2 Técnicas de clasificación

Como se ha especificado anteriormente, se va a estudiar y comparar el uso de dos técnicas de clasificación [12], que, si bien pueden seguir siendo usadas por ciertos servicios, el uso de técnicas de aprendizaje automático e inteligencia artificial, supone cierta mejora con respecto a estas técnicas:

- Número de puerto: Esta técnica se basa en usar el número de puerto de las distintas aplicaciones para que estas puedan ser clasificadas. Existen multitud de aplicaciones o protocolos que tienen un puerto fijo e identificable. Estos son el caso por ejemplo de los protocolos DNS (puerto estándar 53), FTP (normalmente puerto 20 y 21), SMTP (puerto estándar TCP 25), etc.
  Como luego se podrá apreciar, en la fase de creación de árboles de clasificación de aplicaciones, el número de puerto es un parámetro muy útil para la clasificación, ya que, al ser un valor fijo para muchos tipos de aplicaciones, al clasificador le resulta muy sencillo clasificar mediante este parámetro y dar muy buenos resultados (algo que se comprobará más adelante).

  Sin embargo, este método ya no es del todo efectivo, ya que existen y a lo largo del tiempo han surgido numerosas aplicaciones que no poseen un puerto fijo y que, al ser variable, no puede ser clasificado de esta forma. Incluso existen aplicaciones que pretenden ser ocultas a la hora de ser clasificadas como tráfico y ocultan este número de puerto para, así, no ser detectadas de forma tan sencilla.
  Es cierto que este tipo de clasificadores son óptimos en cuanto al uso de memoria, latencia y rapidez a la hora de obtener resultados ya que simplemente se guían por un valor numérico. Sin embargo, existen problemas para los que el uso de esta técnica no da soluciones.
  El caso de la privacidad, como se ha comentado anteriormente, es un problema que afecta a este método, ya que muchas aplicaciones prefieren ser ocultas, cuanto mayor sea posible, lo que incluye ocultar el número de puerto a ser posible.

Puerto	Protocolo	Servicio
22	tcp	ssh
9929	tcp	nping-echo
80	tcp	http
31337	tcp	Elite
23	tcp	telnet
139	tcp	netbios-ssn
6699	tcp	napster
445	tcp	microsoft-ds
6346	tcp	gnutella
21	tcp	ftp
4662	tcp	edonkey
2323	tcp	3d-nfsd

Figura 2-1: Identificación de servicio mediante el número de puerto (herramienta *Zenmap*)

- *Deep Packet Inspection (DPI)*: Este método [10] se trata de una técnica mucho más avanzada que el caso anterior. Esta técnica clasifica paquetes, filtrándolos según el contenido de una parte específica de datos y de su carga útil. El analizar esta carga útil proporciona información que, mediante técnicas como el uso de números de puertos, no podría ser accesible de otra forma. Sin lugar a dudas esta información es de gran utilidad a la hora de elaborar y clasificar tráfico de una forma más óptima (aunque también incluye una serie de problemas que se comentarán más adelante).

  Esta técnica puede ser muy utilizada en el ámbito de la seguridad, para filtrar posibles ataques maliciosos.
  Se necesita de un poder computacional mucho más alto que métodos que solo se basan en analizar parámetros de la cabecera de un paquete, y en un principio los clasificadores no disponían de la tecnología suficiente para poder analizar la carga útil de una gran cantidad de paquetes y proporcionar resultados en tiempo real.
  Es mucho más sencillo, observando el contenido de los paquetes, definir de que tipo de aplicación se trata.

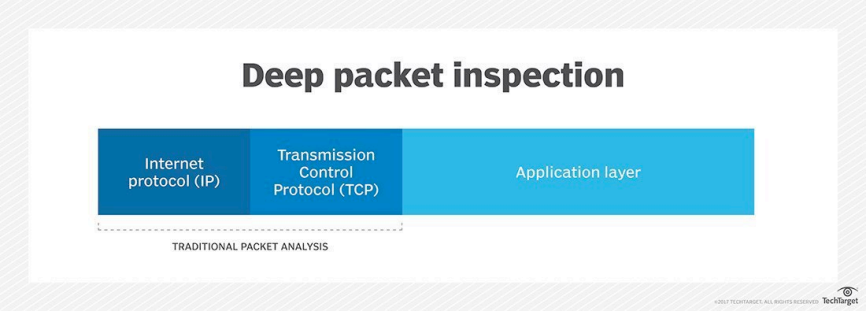

Figura 2-2: *Deep Packet Inspection [10]*

Como se ha comentado anteriormente, el uso de esta técnica está muy relacionado con la seguridad informática y con la detención de paquetes potencialmente peligrosos. Pero también está muy relacionada con el ordenamiento y organización de redes. Una vez analizada la carga útil y el contenido de un paquete, existe la posibilidad de poder

priorizar, dependiendo de los elementos analizados, este paquete o que pase por otros determinados filtros y destinatarios, dependiendo del contenido del paquete.

Sin embargo, esta técnica, comparada con el uso de técnicas de aprendizaje automático, tiene una serie de limitaciones importantes. En primer lugar, para que este sea efectivo a tiempo real, se requiere un gran consumo de recursos de memoria y se necesita una actualización constante de las aplicaciones que dependen de los resultados de este tipo de técnica.

Pero sin duda, uno de los problemas existentes más importantes, es el problema de la privacidad. En el anterior punto se ha comentado que existen aplicaciones que pretenden permanecer ocultas a la hora de ser clasificadas, ocultando incluso el número de puerto.
Por lo tanto, existen muchos más problemas de privacidad, si se pretende acceder, de forma casi total y libre, a los datos de cada uno de los paquetes. Esta información puede ser mucho más sensible y no cualquier empresa debe tener acceso a ella.

Por lo que estos problemas hacen necesaria la búsqueda de un método mucho más efectivo que combine las ventajas de los dos métodos anteriormente descritos, la rapidez y escaso coste computacional y la ausencia de acceso a información que podría ser sensible y privada.

Este método será el usado en el presente trabajo, las técnicas de aprendizaje automático. Producirá resultados óptimos, sin tener que acceder a información que podría estar oculta o que compromete la privacidad de los flujos recogidos (en los conjuntos de datos que se usarán en el presente trabajo, y que más adelante se describirán, no se encuentra ningún tipo de parámetro o información que pueda identificar a emisores o receptores de cada uno de los flujos, como podrían ser las direcciones *IP).*

2.3 Uso de flujo de paquetes

Durante esta parte, se va a justificar el hecho de haber utilizado conjuntos de datos compuestos por flujos de paquetes. Aunque más adelante se comentará acerca de los parámetros que componen cada uno de los flujos, en el caso de los conjuntos de datos utilizados (cedidos por la UPC, como se comentará más adelante), se realizará también una breve descripción de los flujos de datos y el beneficio que supone usarlos.

Mucho más adelante, en el cuarto capítulo, donde se realiza un estudio sobre la posibilidad de clasificación de malware, se necesitará extraer los flujos de tráfico malicioso para poder así generar un árbol de clasificación en base a ellos. Para ello es necesario un pleno entendimiento de los flujos de tráfico.
Se puede definir un flujo como el conjunto de varios paquetes dentro de una, llamado de forma coloquial, conversación. En esta conversación existen características comunes en todos los paquetes. Como pueden ser las direcciones *IP* de origen y destino, así como los puertos de origen y destino.
Estos son los parámetros por el que, mediante un script que más adelante se comentará, se podrá realizar una extracción automática de los flujos pertenecientes a una captura de tráfico cualquiera.

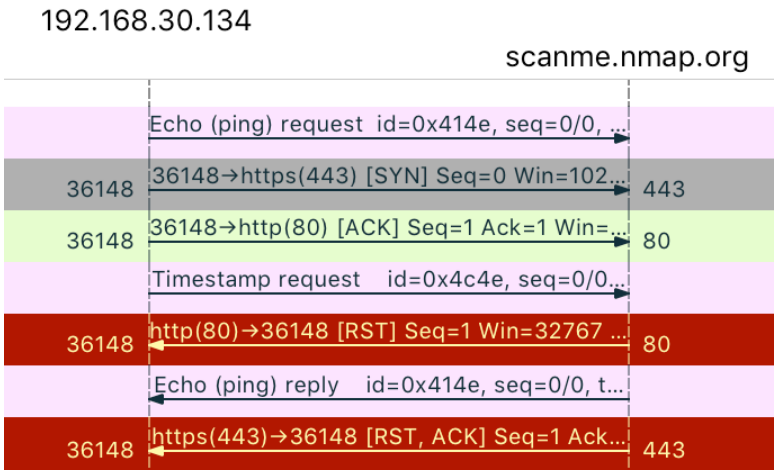

Figura 2-3: ***Ejemplo de diagrama de flujo***

Como veremos más adelante pueden existir incluso flujos de un solo paquete.
Es por este tipo de información por lo que es preferible el uso de flujos y no de paquetes individuales, y es que disponemos de información extra que resulta de gran utilidad a la hora de crear árboles de clasificación óptimos. Un caso, que se observará más adelante en el cual este tipo de información es necesaria, es en el caso del tráfico malicioso. Un ejemplo sencillo es el de los ataques de denegación de servicio, en los cuales una característica fundamental es que consisten en flujos de un solo paquete, algo muy útil (a parte de otros parámetros, como la duración de cada flujo) para poder ser clasificados con una alta precisión.

2.4 Conclusiones

Durante esta fase del estudio del estado del arte se ha comentado acerca de métodos de clasificación de tráfico de internet, como lo son la clasificación mediante el número de puerto, así como la técnica *DPI*. Ambas técnicas tienen sus ventajas y desventajas, y aunque pueden resultar útiles para ciertas aplicaciones, para el caso concreto del trabajo que se expone, podemos considerar que las técnicas de aprendizaje automático y más concretamente el uso y generación de árboles de clasificación es la técnica que mejor se ajusta a las necesidades de este trabajo.

Debido a que no disponemos de información ni la carga útil de los flujos que se nos proporcionan para la realización de este trabajo, no es posible aplicar métodos que se basen en inspeccionar la información de cada paquete, si no que debemos analizar los flujos, basándonos en datos más estadísticos (que luego se enumerarán y describirán) o datos que no suponen ningún problema para la privacidad.

Por ello es muy beneficioso el uso de técnicas de aprendizaje automático. A pesar de contar con datos casi puramente estadísticos, sin ninguna información más profunda sobre el contenido de cada paquete, realizando una correcta etapa de entrenamiento y etapa de test y habiendo configurado el clasificador de la manera más óptima posible, se pueden obtener resultados muy satisfactorios, que sin duda se observarán más adelante.

Por último, se han estudiado los beneficios, para la construcción de un clasificador, del uso de flujos de paquetes, tanto para la construcción de un árbol de clasificación, como para su etapa de testeo. Es por ello por lo que se ha decidido (y por qué es más sencillo adaptarlo

con los conjuntos de datos que han sido cedidos para la realización de este trabajo) usar flujos de datos para la clasificación de aplicaciones y tráfico malicioso más adelante.

3 Diseño

3.1 Introducción

Durante esta fase de diseño se estudiarán, tanto los conjuntos de datos que se van a usar para la clasificación, como una parametrización del entorno Weka que servirá de mucha utilidad para poder filtrar campos de estos conjuntos de datos de una manera muy gráfica y sencilla y que será necesario para construir los árboles de clasificación.
El esquema general que seguirá el capítulo es el siguiente:

- Procesado de los conjuntos de datos y filtrado
- Parametrización del software Weka

3.2 Procesado de los conjuntos de datos

Para poder poner en funcionamiento el clasificador, antes en esta fase se adaptarán los conjuntos de datos a un formato especifico que Weka necesita para realizar una correcta fase de entrenamiento y test del clasificador, ya que, sin este formato, Weka directamente no abre el conjunto de datos y no se puede utilizar para elaborar un árbol de clasificación ni poder filtrar campos de este mismo conjunto de datos.

Uno de los problemas de esta fase es el gran tamaño de los conjuntos de datos y es que hay alguno que alcanza la cifra de los nueve millones de flujos. Este gran número de flujos influye en el tiempo necesario para realizar tanto la construcción de un clasificador como para el testeo del mismo como se observará en la siguiente fase. Será necesario modificar el tamaño de memoria que necesita Weka para poder funcionar para que así la clasificación de conjuntos de datos tan grandes sea posible.

3.2.1 Descripción de los datos a usar (UPC)

Los conjuntos de datos que se usarán durante todas las pruebas de clasificación han sido cedidos por la Universidad Politécnica de Cataluña (UPC), la cual conecta 25 facultades y 40 departamentos a internet a través de la red española para interconexión de los recursos informáticos de las universidades y centros de investigación.
En total se dispondrán de 7 conjuntos de datos distintos tomados desde el día 11-12-08 hasta el 10-3-09, de los cuales se usarán 6 en este estudio.

Conjuntos de datos usados	Número flujos	Fecha captura	Hora captura	Duración
UPC-1	2.985.098	11/12/08	10:00	15 min
UPC-2	3.369.105	11/12/08	12:00	15 min
UPC-3	3.474.603	12/12/08	16:00	15 min
UPC-4	3.020.114	12/12/08	18:30	15 min
UPC-5	7.146.336	21/12/08	16:00	1 hora
UPC-7	5.510.999	10/3/09	3:00	1 hora

Figura 3-1: Conjuntos de datos UPC

Como se puede observar, los datos fueron tomados a distintas horas del día y los conjuntos 1,2,3 y 4 tienen una duración de 15 min de tiempo en el que flujos de la universidad son capturados y los conjuntos 5 y 7 tienen una duración de una hora.

Durante este trabajo y más adelante se estudiará la influencia del clasificador y el porcentaje de muestras clasificadas correctamente al existir mayor dependencia entre conjuntos de datos. Esto puede producirse al tomar conjuntos de datos de periodos de tiempo cercano. En el caso de los datos de la UPC se dispone de datos en periodos de tiempo cercanos (conjuntos 1,2,3 y 4 con una diferencia de menos de 24 horas) como en periodos lejanos (conjunto 7 tomado varios meses más tarde que el resto).

Para poder realizar un correcto clasificador basándonos en los conjuntos de datos de la UPC, es necesario entender los campos de estos conjuntos de datos.
Estos campos son datos estadísticos que no incluyen direcciones IP para así proteger la privacidad de cada uno de los flujos. La aplicación definida en cada uno de los flujos ha sido obtenida previamente por la UPC a partir del resto de campos mediante un sistema L7-Filter. Los campos de los conjuntos de datos serían los siguientes:

Pr	SrcP	DstP	Pkts	Octets	StartTime	EndTime	Active	B/Pk	Ts	Fl	Application
06	50	114f	2	3000	0901.00:59:15.924	0901.00:59:17.924	2.000	1500	00	10	skypetoskype

El primer parámetro es el número de protocolo, siendo 6 para protocolo TCP y 17 para UDP. El segundo parámetro es el puerto de origen. El tercer parámetro es el puerto de destino. El cuarto parámetro es en el número de paquetes dentro de flujo. El quinto parámetro es el número de bytes de ese flujo. El sexto y el séptimo parámetro son los tiempos de inicio y final de ese flujo dentro de la captura de datos tomada. El octavo parámetro es la duración de ese flujo (resta entre tiempo final y tiempo inicial). El noveno es la división del número de Bytes del flujo entre el número de paquetes totales en ese mismo flujo. Por último, antes de los flags TCP, se encuentra el campo del *Type of Service*. A continuación, se encuentran una serie de campos que son flags TCP (campos a 0 en el caso de que se trate de un flujo UDP). Los flags TCP que se encuentran entre los parámetros son: URG, ACK, PSH, RST, SYN, FIN. Como se observará en la fase de desarrollo de los árboles de clasificación de aplicaciones vamos a filtrar varios de estos campos para sí encontrar el modo de clasificación más óptimo.

Como se puede observar, en los conjuntos de datos no hay ninguna información acerca de las direcciones IP por motivos de privacidad de los usuarios. Esto supone una gran ventaja respecto a las técnicas DPI anteriormente descritas.

3.2.2 Puesta en funcionamiento del programa adaptando los conjuntos de la UPC

Existe un tipo de formato específico admitido por el programa Weka. Este formato es el llamado *Atribute-Relation File Format* [5] (Durante este trabajo se referirá a él como ARFF). Un archivo ARFF es un archivo de texto ASCII que describe una lista de instancias que comparten un conjunto de atributos. Los archivos ARFF fueron desarrollados por el Departamento de Ciencia de Computación de la Universidad de Waikato para su uso en el software de aprendizaje automático Weka.

Los archivos pertenecientes a este tipo de formato siguen siempre la misma estructura.
Se tiene una cabecera en la quedarán escritos (se podría decir inicializados) todos los atributos que contienen los datos. (Se definirán los parámetros de los que se componen los flujos en los conjuntos de datos)

```
@attribute 'Pr' real
@attribute 'SrcP' real
@attribute 'DstP' real
@attribute 'Pkts' real
@attribute 'Octets' real
@attribute 'StartTime' real
@attribute 'EndTime' real
@attribute 'Active' real
@attribute 'Bytes_packet' real
@attribute 'ToS' real
@attribute 'URG' real
@attribute 'ACK' real
@attribute 'PSH' real
@attribute 'RST' real
@attribute 'SYN' real
@attribute 'FIN' real
```

Figura 3-2: Atributos formato ARFF

Simplemente se definirán con un nombre los atributos para indicar al software Weka de que parámetros se componen cada uno de los flujos. Más tarde mediante esos nombres se podrán filtrar cada uno de los campos.

Durante esta fase surgirá uno de los problemas. En el formato ARFF se tendrá que definir el parámetro "aplicación" y en el escribir todas las aplicaciones que van a encontrarse dentro de cada conjunto de test en cada flujo. Evidentemente, no se puede extraer de cada flujo manualmente que aplicaciones se han detectado y escribir dentro del atributo "app" una lista de todas las aplicaciones.

Para solucionar este problema y automatizar este proceso, se ha desarrollado un script en *Python* que, dado un conjunto de datos como los proporcionados por la UPC, extrae las aplicaciones definidas en el último parámetro de cada flujo y elabora una lista de aplicaciones de manera que queden todas enumeradas y sin repetirse. Finalmente se copia esta lista en un nuevo atributo en la cabecera del archivo ARFF denominado *app*.

@attribute 'app' {ssl, http, bittorrent}

Tras haber definido la cabecera, quedaría definir el cuerpo del archivo ARFF. Esto es un trabajo mucho más simple ya que simplemente se escribiría a continuación de la cabecera: *@DATA* y tras un salto de línea se copian todos los flujos extraídos de los conjuntos de datos de la UPC de la siguiente forma:

@data
6,80,1495,6,1803,1228986008.637849,1228986008.712956,0.075107,300,0,0,1,1,0,1,1,unknown
6,1495,80,5,788,1228986008.634605,1228986008.708957,0.074352,157,0,0,1,1,0,1,1,unknown
6,1231,80,4,736,1228986008.639200,1228986008.692974,0.053774,184,0,0,1,1,0,0,1,unknown
6,80,1231,5,388,1228986008.639106,1228986008.735524,0.096418,77,0,0,1,1,0,1,1,unknown
6,52918,4004,4,333,1228986008.657748,1228986008.711035,0.053287,83,0,0,1,1,0,0,1,edonkey

De esta forma ya se dispone de los conjuntos de datos de la UPC convertidos al formato ARFF para que de esta forma puedan ser procesados por el software Weka.

3.3 Parametrización del software Weka

El entorno Weka es un paquete de software libre que incluye una serie de algoritmos de aprendizaje automático que será una herramienta fundamental en todo este trabajo. Se usará para poder elaborar un clasificador con los conjuntos de datos y también para trabajar con estos conjuntos de datos, filtrando campos de los mismos.

3.3.1 Descripción del algoritmo a usar y sobre árboles de clasificación

En el caso de este trabajo, se centrará en un tipo de algoritmos para la clasificación, los árboles de clasificación. Se partirán de los resultados obtenidos en la elaboración de un trabajo de fin de máster [4], en el cual se obtiene que los árboles de clasificación creados mediante el algoritmo J48 da resultados muy satisfactorios en comparación con el resto de algoritmos disponibles en el software Weka.

Es difícil determinar el comienzo de lo que conocemos por árboles de clasificación. En 1950, se realizó una publicación de "Concept Learning Systems" en la cual se construían árboles de clasificación inducidos a partir de un conjunto de ejemplos.

A mediados de los 80 un grupo de estadísticos publicaron el libro "Classsification and regression trees" en el que se presentaba un método para construir un clasificador de forma recursiva y que se representaba como un árbol(CART). En paralelo a este trabajo, se desarrolló el C4.5, evolución de un programa previo del mismo autor denominado ID3. Ambos métodos, CART y C4.5 son muy diferentes pero los resultados de clasificación a los que llegan son muy similares.

El tipo de árbol que usa el software Weka es J48 que es un tipo creado por ellos mismos, y que a su vez es una evolución del C4.5.

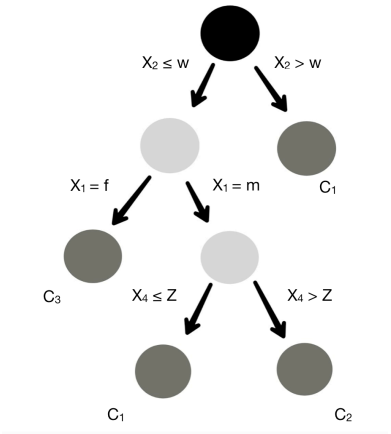

Figura 3-3: Esquema árboles de clasificación [8]

Como se puede observar en la figura 3-2, todo árbol de clasificación comienza con un nodo al que pertenecen todos los casos de la muestra (nodo negro). El resto de nodos pueden ser nodos no terminales o intermedios (nodos grises claros) y nodos terminales u hojas (nodos

grises oscuros).

En la construcción del árbol cada nodo hoja corresponde a una categoría concreta de la variable clase. De esta forma los nodos hoja representan las particiones en las que se ha dividido el espacio de clasificación. Al número de nodos hoja del árbol se le denomina complejidad del árbol. En el caso de la figura 3-2 es 4.

Uno de los motivos por lo que se usan este tipo de árboles de clasificación es debido a que aportan una explicación sobre esta clasificación. Para avanzar desde el nodo raíz hasta los nodos terminales tenemos una serie de condiciones que quedan reflejadas en el árbol lo cual nos da una explicación de cómo se llega a esa clasificación. Esto que las redes neuronales no son capaces de proporcionar. Este aspecto es muy importante en ámbitos como medicina, detención de fraude o marketing.

3.3.2 Clasificación y filtrado de datos por Weka

En esta sección se explicará cómo se va a realizar la creación de un modelo de clasificación y el filtrado de los parámetros de los conjuntos de datos con el programa Weka. Sin embargo, aquí no se realizará ninguna prueba sobre ningún conjunto de datos ya que esto se realizará en la siguiente fase.
Dentro del software Weka existen tres opciones distintas de tipos de simulación que se evaluarán [13]:

- Utilizar un conjunto de datos para la etapa de entrenamiento (construcción del árbol de clasificación) y otro conjunto distinto para la etapa de test.
- Utilizar un mismo conjunto de datos para las dos etapas, dividiendo un porcentaje del mismo para la etapa de entrenamiento y el resto para la de test.
- Simulación similar al anterior caso, pero este porcentaje definido para realizar la etapa de test irá variando en diferentes iteraciones (validación cruzada).

En ellas se puede elegir el modo de realizar la clasificación y como funcionarán tanto la etapa de entrenamiento como de test. Y es que durante la construcción del clasificador se realizan dos fases. La primera de ellas, la fase de entrenamiento, en la cual a partir del conjunto de datos se construirá un árbol de clasificación. La segunda de estas fases es la fase de test que, como su nombre indica, testea el árbol creado en la anterior etapa y proporciona amplios resultados sobre este testeo.

3.4 Conclusiones

Como se ha demostrado en este capítulo de diseño, es necesario adaptar los conjuntos de datos cedidos por la UPC al formato que acepte Weka. Esto requiere entender de forma correcta este formato y el desarrollo de herramientas para que la adaptación de conjuntos de grandes dimensiones sea posible.

De forma adicional, para el siguiente capítulo, será necesario entender cómo funciona Weka, el cual, incluye multitud de opciones para poder realizar ajustes previos a la clasificación, así como una gran cantidad de opciones también de modos de clasificación (tanto para la etapa de entrenamiento como de test).

En el capítulo siguiente, y una vez entendido como funciona el programa, se realizarán diversas pruebas sobre los conjuntos de entrenamientos para encontrar el clasificador que proporcione los mejores resultados posibles.

4 Desarrollo de árboles de clasificación de flujos de tráfico

4.1 Introducción

Una vez entendido el funcionamiento del programa Weka y de toda su funcionalidad existente, se aprovechará para el desarrollo de árboles de clasificación de aplicaciones.
Como se ha comentado en el capítulo anterior, se van a utilizar los conjuntos de datos recogidos por la UPC para así poder realizar un clasificador que se adapte lo más posible al tráfico real y poder clasificar aplicaciones con un alto porcentaje de aciertos.

Durante esta fase se va a realizar un gran número de pruebas:

- **Primer tipo de filtrado**:
  - Realización de todas las simulaciones posibles en el programa Weka
  - Conclusión de qué caso de simulación da mejores resultados
- **Segundo tipo de filtrado**: Realizando la simulación que ha dado mejores resultados en el anterior tipo
- **Tercer tipo de filtrado**: Una vez más, con la simulación que ha dado mejores resultados en el primer tipo de filtrado
- **Resultados obtenidos**: Se analizarán los resultados obtenidos y especialmente se estudiarán los casos de los flujos que poseen una mayor precisión de clasificación, una menor precisión y se analizará una aplicación concreta de los resultados obtenidos, la clasificación de flujos de tráfico malicioso.

Comentar que durante este capítulo se hará frente a una serie de problemas que es la, comentada anteriormente, gran cantidad de flujos dentro de los conjuntos de datos. Esto provoca que para cada uno de los casos realizados se haya requerido un gran tiempo de simulación (juntando la etapa de entrenamiento y la etapa de test) y que el terminal desde el que se han realizado las pruebas dedicara todos sus recursos a ello.

4.2 Primer tipo de filtrado

Las pruebas comenzarán filtrando una serie de campos dentro del conjunto de datos. En este primer intento, se filtrarán campos considerados, en una primera instancia, sin importancia dentro de la simulación para la clasificación de aplicaciones o que son campos que a su vez dependen de otros campos.

De los 17 parámetros o atributos que tienen los conjuntos de datos, se filtrarán los siguientes:

- En primer lugar, el llamado *EndTime*. Este parámetro depende de otros dos que son el *StartTime* y el tiempo activo que dura el flujo, por lo que, teniendo ya estos dos últimos, en esta primera prueba se considera que no es necesario el parámetro que marcaba el tiempo en el cual un flujo terminaba.

- El siguiente parámetro filtrado fue el que define el número de bytes por paquete. Ocurre de forma similar con el caso anterior, y es que ya existen otros dos parámetros que son, el número de bytes en total del flujo y también el número de paquetes enviados dentro de un flujo, por lo que el atributo bytes por paquete no es más que una división de los dos anteriores.

- El tercer parámetro filtrado fue el que marca el *Type of Service (ToS)*. Como este parámetro suele tomar el valor de 0 que significa que no posee prioridad, consideré adecuado filtrarlo (su valor no cambia en los conjuntos de datos).

- El cuarto parámetro filtrado es el flag TCP *URG*. Este flag marca un paquete como urgente, y como a simple vista no parecía variar ni ser de vital importancia dentro de los conjuntos de datos, decidí proceder con su filtrado y así además reducir de forma teórica el tiempo necesario para realizar las simulaciones (cuantos menos atributos existan, menos tiempo será necesario para construir un árbol de clasificación en torno a ellos).
- Por último, y de forma similar al caso anterior, también se realiza el filtrado del flag *RST* que aborta la conexión.

El resto de flags TCP permanecen sin haber sido filtrados.

Figura 4-1: Primer filtrado

Una vez realizado el primer tipo de filtrado, se realizarán varios casos de simulaciones distintas, hasta encontrar la forma en la cual se obtienen los mejores resultados. Se han utilizado todas las opciones disponibles en el programa Weka.
Las salidas proporcionadas por el programa Weka representadas en el anexo C.

Descripción de los distintos casos:

- **Primer caso**: se dividirá el conjunto UPC-1 en un 90 % para la etapa de entrenamiento y un 10 % para la etapa de test.
- **Segundo caso**: se dividirá el conjunto UPC-1 en un 99 % para la etapa de entrenamiento y un 1 % para la etapa de test.
- **Tercer caso**: Validación cruzada, al igual que en el primer caso, utilizaremos un 10% del conjunto para la etapa de test, pero se irá variando este 10% en varias iteraciones.
- **Cuarto caso:** Se utilizará el conjunto UPC-1 para la etapa de entrenamiento y, para testear el árbol de clasificación creado, otro conjunto de datos. (UPC-2, UPC-3, UPC-4, UPC-5, UPC-7).

Los resultados de los diferentes casos y tipos de filtrados estarán adjuntos en el punto 4.5.

De forma adicional, comentar que, tras los dos primeros casos de simulaciones y viendo que los resultados obtenidos son casi perfectos, nos enfrentamos a un problema, el problema del sobreajuste [7].
El árbol de clasificación ha sido construido mediante un conjunto de datos, el cual, ha sido el mismo que se ha utilizado durante la etapa de test. Esta práctica es un tanto peligrosa ya que, el árbol no ha sido testeado con datos semejantes a la realidad, ha sido testeado con los mimos datos con los que se ha construido ese árbol por lo que los resultados siempre van a ser satisfactorios

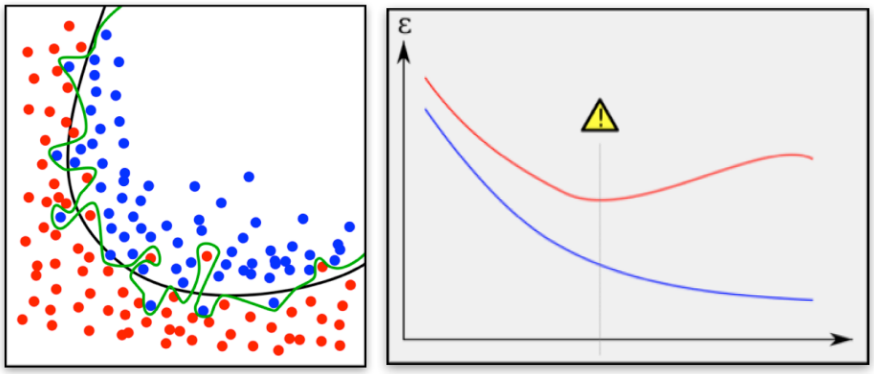

Figura 4-2: Sobreajuste

Para solucionar este problema se realizarán los siguientes casos de simulaciones.

4.3 Segundo tipo de filtrado

Con el fin de mejorar los resultados obtenidos en el caso anterior y a su vez, realizar un filtrado más realista, se filtrarán los siguientes parámetros:

Protocolo
Puerto origen
Puerto destino
Número paquetes
Bytes
Tiempo inicio
Tiempo final
Tiempo activo
bytes/paquete
ToS
URG
ACK
PSH
RST
SYN
FIN

Figura 4-3: Segundo tipo filtrado

En este tipo de filtrado sólo se filtrarán campos que se pueden obtener dependiendo de otros parámetros. Estos son el tiempo de inicio de un flujo *(StartTime)*, el tiempo de

finalización de un flujo *(EndTime*) y el parámetro que define el número de bytes por paquete dentro de un flujo.

Para este tipo de filtrado se realizará una simulación en la cual, se utilizará el primer conjunto de datos para la construcción del árbol de clasificación y para la etapa de entrenamiento y el quinto conjunto de datos para la validación y la etapa de testeo (técnica que se ha visto en el anterior tipo de filtrado, es la más eficaz y mayor porcentaje de muestras clasificadas correctamente proporciona).

4.4 Tercer tipo de filtrado

Se ha comprobado que, realizando un segundo tipo de filtrado, aunque los resultados son satisfactorios, no se han mejorado respecto al mismo tipo de simulación con el primer tipo de filtrado (séptimo caso).

Por lo tanto, a continuación, se va a realizar un tercer tipo de filtrado para estudiar si se puede mejorar los resultados obtenidos. Al igual que en el tipo anterior, se realizará el mismo tipo de simulación, utilizando el primer conjunto de datos para construir el árbol de clasificación y el quinto conjunto de datos para verificar el funcionamiento de ese árbol.

En el anterior caso de filtrado, se habían eliminado tres parámetros de los flujos que dependían de otros parámetros. Estos eran el *StartTime*, el EndTime y el número de bytes por paquete.

De forma adicional, en este tercer tipo de filtrado, se eliminará el campo *Type of Service* (*Tos*). Este campo no es muy útil en la clasificación de aplicaciones ya que no es recomendable guiarse por el valor de un solo bit para decidir y clasificar una aplicación. Mediante este hecho, se podría hacer referencia al famoso RFC del 1 de abril de 2003 [16] (famoso Request for Coments de carácter humorístico debido al día de los inocentes, 1 de abril). En el cual se definía un bit llamado “Evil bit” con el cual se podría detectar ataques simplemente observando si ese bit estaba activado.

Figura 4-4: Tercer filtrado

4.5 Resultados obtenidos

Durante este apartado se expondrán una serie de figuras para para la explicación de los resultados obtenidos.

Primero en la figura 4-5 se expondrán el porcentaje de muestras clasificadas correctamente en cada uno de los casos del primer filtrado.

	Caso 1	Caso 2	Caso 3	UPC-2	UPC-3	UPC-4	UPC-5	UPC-7
Porcentaje de clasificación	91,5%	91,7%	NULL	85,2%	81,1%	85,8%	86,4%	80,8%

Figura 4-5: Resultados con los distintos conjuntos de datos

Como se ha comentado anteriormente, los resultados de los dos primeros casos no proporcionan una medida totalmente real. Esto es debido a que realizar la fase de testeo con el mismo conjunto de datos que se ha usado para construir el árbol de clasificación puede provocar un sobreajuste y falsos resultados (debido a la total dependencia que existe de los conjuntos de datos).

De forma adicional comentar que, el caso 3 queda definido como NULL ya que esta prueba no se llegó a realizar. Este caso se trata de la ya explicada validación cruzada, explicada en la fase de diseño. Debemos tener en cuenta que, en el primer caso que dividimos el conjunto de la misma forma que en el cuarto caso, la simulación tarda aproximadamente 1 hora y media. Así que, si se tiene que realizar este tipo de simulación variando el porcentaje usado para la etapa de testeo (en unas 10 iteraciones), el tiempo de simulación sería inviable ya que superaría las 10 horas.

Como se puede observar, los mejores resultados se obtienen en el caso del conjunto UPC-5. En este caso se utiliza el primer conjunto para la etapa de entrenamiento y el quinto conjunto para la etapa de test. Esto puede deberse a que existe una gran dependencia entre las muestras del primer y quinto conjunto, a pesar de que existen varios días entre la captura del primer conjunto y el quinto.

Un factor que influye mucho en el porcentaje de muestras clasificadas correctamente es la dependencia de los datos. De esta forma podemos observar que conjuntos como el séptimo dan peores resultados ya que fueron recogidos meses después a las 3 de la mañana (horario en el que el tráfico dentro del campus se presupone diferente a lo habitual).

Entre los factores que pueden influir en que el conjunto UPC-5 de mejores resultados se encuentran:

- Tamaño del conjunto: Tiene un número de flujos significativamente mayor que los anteriores conjuntos.
- Que exista una dependencia de datos mayor: Esto puede deberse a multitud de factores y se necesitaría conocer las condiciones del campus durante los días de captura para evaluarlo. Puede ser que el tráfico fuera similar, o que, debido a que el quinto conjunto tiene un tamaño superior, hay mayor posibilidad de encontrar tráfico parecido.

Existen multitud de variables para evaluar porque se obtienen mejores resultados con el quinto conjunto.

A continuación, en la figura 4-6, se expondrán los resultados al utilizar los diferentes conjuntos de datos de la UPC.

	UPC-2	UPC-3	UPC-4	UPC-5	UPC-7
Porcentaje de clasificación	85,1%	81,1%	85,8%	86,3%	80,8%

Figura 4-6: Resultados con los distintos conjuntos de datos aplicando UPC-1 como conjunto de entrenamiento

En las simulaciones realizadas durante el primer tipo de filtrado, hemos visto los resultados obtenidos al utilizar diversos conjuntos de datos para verificar el modelo creado hasta comprobar cuál es el resultado más satisfactorio para aplicarlo a los siguientes tipos de filtrado.

Por último, tras realizar todas las simulaciones en el primer tipo de filtrado, realizamos dos tipos de filtrado más, para poder mejorar los resultados obtenidos. Sus resultados están adjuntos en la figura 4-7.

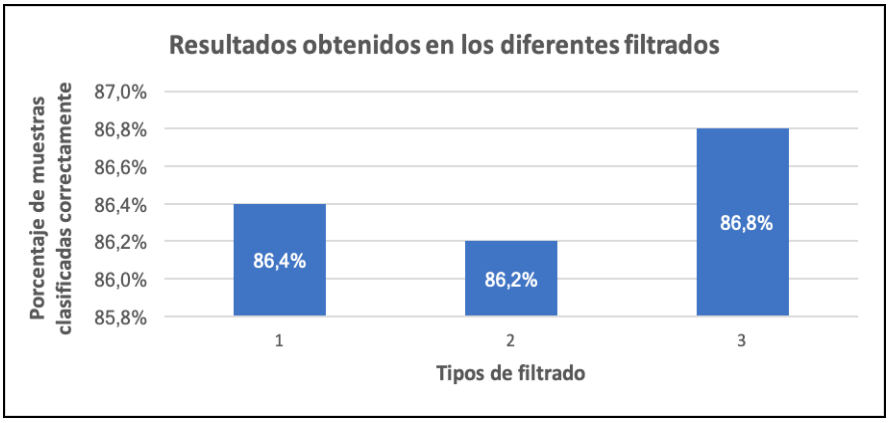

Figura 4-7: Resultados todos los filtrados

Los mejores resultados se han obtenido con el tercer tipo de filtrado. En este tipo se filtran campos que a su vez depende de otros campos (Tiempo de inicio o final y el campo de *Bytes/packet*). También se filtra el campo *ToS*, ya que el clasificador no se puede guiar por un simple bit que puede ser fácilmente modificado y manipulado y puede dar a una errónea interpretación de por qué se llega a clasificar una aplicación de un modo u otro (es preferible que el clasificador se guíe por medidas más estadísticas como lo son el número de paquetes dentro de un flujo o el tiempo activo de ese flujo).

Comentar que, finalmente y gracias a los conjuntos de datos proporcionados por la UPC, se han obtenido resultados muy satisfactorios. Al final obtenemos un porcentaje de muestras clasificadas correctamente de un 87% aproximadamente, mediante la construcción del modelo con el primer conjunto de datos y la verificación y la etapa de test con el quinto conjunto de datos.

Adicionalmente, el programa Weka, nos proporciona más resultados y conjuntos de medidas.

Uno de estos resultados es una matriz de confusión (debido a que existen un gran número de aplicaciones a clasificar, la matriz de confusión también será de un gran tamaño) para poder detectar en qué aplicaciones ha existido un mayor problema a la hora de ser clasificadas.

Por último, podemos analizar medidas individuales para cada aplicación clasificada. En el caso del presente trabajo primero analizaremos las aplicaciones que son clasificadas con mayor precisión, seguidas de las que poseen una menor precisión y terminaremos con una aplicación concreta para este tipo de resultados, la clasificación de tráfico malicioso.

4.5.1 Flujos clasificados con mayor precisión

Como resultado, se va a obtener que las aplicaciones con una mayor precisión a la hora de ser clasificadas, son aquellas que poseen un número de puerto fijo, ya que el clasificador puede diferenciarlas del resto mediante este número de puerto.

Varias de las aplicaciones que poseen una alta precisión en clasificación son las siguientes:

- FTP: Este tipo de servicio posee un puerto fijo que es el puerto 20, y cómo podemos observar en la siguiente imagen (imagen del árbol resultante creado por el clasificador), el clasificador va reduciendo el puerto de destino hasta que pueda filtrar este tipo de aplicación.

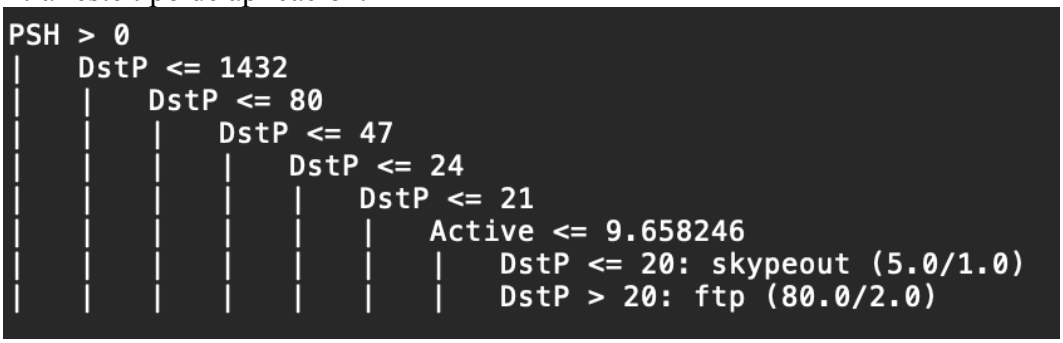

Figura 4-8: Árbol de clasificación de FTP

- SMTP: Esta es otra de las aplicaciones que poseen un puerto fijo, en este caso el puerto 25, por lo tanto, va a proporcionar igualmente una alta precisión.

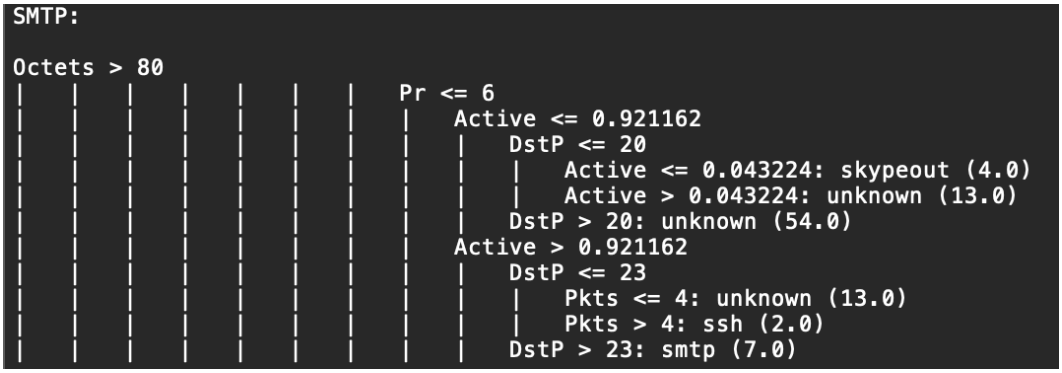

Figura 4-9: Árbol de clasificación de SMTP

Como podemos observar en su árbol de clasificación, finalmente en el último filtrado, el árbol de clasificación se guía por el numero de puerto (mayor que 23, es el 25), pero tambien decide en torno a otros parámetros, es por ello por lo que

tendrá una mayor precisión que en el caso anterior, FTP. Entre estos parámetros se ecnuentran el tiempo de duración del flujo así como la cantidad de bytes, y por supuesto por el protocolo, que es TCP.

- DNS: Por último, esta última aplicación que se va a analizar individualmente, también tiene un numero de puerto fijo que es el 53.

```
DNS:

DstP > 52
|   |   |   Active <= 0.203961
|   |   |   |   Octets <= 50
|   |   |   |   |   Octets <= 46: unknown (99.0/1.0)
|   |   |   |   |   Octets > 46
|   |   |   |   |   |   Octets <= 49
|   |   |   |   |   |   |   SrcP <= 45831: skypeout (2.0)
|   |   |   |   |   |   |   SrcP > 45831: skypetoskype (2.0)
|   |   |   |   |   |   Octets > 49: unknown (15.0)
|   |   |   |   Octets > 50
|   |   |   |   |   Octets <= 96
|   |   |   |   |   |   SrcP <= 845
|   |   |   |   |   |   |   SrcP <= 88: dns (235.0/2.0)
```

Figura 4-10: Árbol de clasificación de DNS

El clasificador, desde el primer momento, ya filtra por un puerto cercano al puerto de la aplicación DNS y, después, filtra este tipo de aplicación según otros parámetros estadísticos. El tráfico DNS es un tipo de tráfico muy identificabl y detectable por lo que es natural que su precisión sea alta. Siempre se trata de flujos de un solo paquete (contiene la peticion DNS) y con pocos bytes por flujo por lo que esta informacion tambien es analizada y utilizada por el clasificador para diferenciarlo del resto de tráfico.

Por último comentar que, existe dentro de los conjuntos de datos utilizados para realizar este árbol de clasificación una gran cantidad de flujos de tráfico DNS, lo que hace que el clasificador filtre y se guíe desde un principio por los parámetros (como el puerto 53) para poder diferenciar antes el tráfico que es más abundante. Esta es otra razón por la cual el tráfico DNS tiene una muy alta precisión.

Se ha realizado un script que lee el resultado obtenido al testear el árbol de clasificación en Weka y extraemos los diferentes valores de precisión para cada aplicación individualmente.

Precisión	Aplicación
0,95	*http*
0,92	*bittorrent*
0,95	*ssl*
0,97	*ftp*
0,98	*smtp*
0,99	*ssh*
0,92	*gnutella*
0,98	*pop3*
0,96	*ntp*
0,98	*dns*
0,91	*socks*
0,98	*quake-halflife*

Figura 4-11: Aplicaciones con precisión más alta

4.5.2 Flujos clasificados con menor precisión

Varias de las aplicaciones que poseen una muy baja precisión en clasificación son las siguientes:

Precisión	Aplicación
0,34	*msnmessenger*
0,32	*jabber*
0,05	*httpvideo*
0,33	*httpcachemiss*
0,99	*httpcachehit*
0,30	*soulseek*
0,03	*ares*
0,25	*x11*
0,35	*mohaa*
0,03	*freenet*

Figura 4-12: Aplicaciones con precisión más baja

En el caso de las aplicaciones con una precisión más baja tenemos claros ejemplos de aplicaciones que pretenden ser indetectables. Es el caso de *freenet* [9], se trata de una red descentralizada que busca el total anonimato de sus usuarios para garantizar una libertad de

expresión, bajo su punto de vista, mayor. Esto unido a que solo existen 4 flujos de este tipo de aplicación en el conjunto de entrenamiento (quinto conjunto de datos), hace que sea un tipo de aplicación con una precisión muy baja a la hora de ser clasificada.

Aplicaciones del tipo *Peer-to-Peer*, como son el caso de Ares y *Soulseek*, también carecen de una alta precisión. Son un tipo de aplicaciones complejas en el que existen múltiples números de puerto y que, en el caso concreto de *Soulseek*, en la matriz de confusión se puede observar que la mayoría del tráfico es confundido con una aplicación *Peer-to-Peer* mucho más conocida y con un mayor número de flujos, *Bittorrent*.

En el caso de las aplicaciones *httpvideo*, *httpcachemiss* y *httpcachehit*, el problema es que el clasificador las clasifica mayoritariamente como tráfico http, ya que este tipo de tráfico es mucho más abundante y el árbol de clasificación se orienta más en torno a este tipo de tráfico.

Por último, comentar el caso del tráfico *Jabber*. Este tráfico es muy importante ya que se trata de un protocolo de mensajería instantánea que ha sido adoptado por empresas muy conocidas del sector tecnológico y usado por millones de personas. Por motivos ajenos a nuestro conocimiento y al tratarse de tráfico exclusivamente del campus (no es un tráfico realmente común) no existen los suficientes flujos de este tipo de aplicación como para que el clasificador pueda construir un árbol que realmente filtre este tipo de tráfico y así se obtenga una alta precisión. Este tráfico es confundido, en la matriz de confusión obtenida con otro tipo de tráfico más abundante, *http* y *bittorrent* en este caso.

4.5.3 Aplicación de los resultados obtenidos, clasificación de flujos de tráfico malicioso

Durante todo este capítulo se han obtenido resultados satisfactorios a la hora de construir un modelo para la clasificación de flujos de tráfico en internet. Gracias a los conjuntos de datos, cedidos por la UPC, se ha observado qué combinación de conjuntos de datos son más efectivos para la generación de un clasificador óptimo.

Durante esta sección se va a utilizar esa combinación para, una vez añadido tráfico malicioso al tráfico normal, construir un clasificador que detecte este tipo de tráfico en internet.

Nos enfrentaremos a diversos problemas. Se deberá obtener tráfico malicioso de forma libre o generarlo. Se tendrá que adaptar esta captura de tráfico malicioso al formato soportado por Weka. Y finalmente se estudiará si el filtrado realizado es suficiente o si es necesario filtrar algún otro campo más para que el clasificador generado sea más óptimo.

Para poder aplicar los resultados obtenidos en una aplicación más concreta como es la detención de tráfico malicioso, se usarán los siguientes ataques (más información acerca de ellos en el anexo):

- Ataque de denegación de servicio (Obtenido de forma libre en internet), etiquetado con el nombre de *malware-ddos* en la realización de las pruebas
- Ataque *Heartbleed* (Obtenido de forma libre en internet), etiquetado con el nombre *malware-heartbleed.*
- Ataque de escaneo de puertos TCP [14] (escaneo nmap generado para la realización de este trabajo) al dominio *scanme.nmap.org*, etiquetado como *nmap-tcp.*

- Ataque de escaneo de puertos TCP (escaneo nmap generado para la realización de este trabajo) al dominio *hackthissite.org*, etiquetado como *nmap-tcp.*

Es de vital importancia el entendimiento de estos dos últimos tipos de ataques, así como el modo en que se han generado.

Este tipo ataques funcionan de la siguiente forma: el atacante envía muchos paquetes con el flag *SYN* activado de muchos puertos distintos, si la dirección *IP* a la que se pretende escanear responde con el flag *SYN/ACK* activo, entonces el puerto está en escucha (abierto). Si por el contrario envía un *RST*, no hay nada escuchando en el puerto. Si no hay respuesta, se envía un error *ICMP*. De esta forma el atacante sabe que puertos están activos y cuáles no. Este tipo de ataque ha sido generado mediante la herramienta *nmap* existente en la distribución *Kali Linux*.

Se ha realizado el mismo tipo de ataques a dos sitios web distintos (por motivos que se detallará más adelante). Estos sitios web son, el dominio *scanme.nmap.org (IP: 45.33.32.156)* y al dominio *hackthissite.org (IP: 137.74.187.102)*.

También se ha usado una herramienta más gráfica, *Zenmap* [15], para poder observar los resultados de este tipo de ataques de forma más gráfica. Comentar, por último, que estos ataques realizados han sido generados en entornos simulados, pero que se asemejan a cómo se comportan en la realidad y los dominios usados como víctimas son dominios especialmente creados para que se puedan realizar pruebas como estas sobre ellos.

	Puerto	Protocolo	Estado	Servicio
	21	tcp	filtered	ftp
	22	tcp	open	ssh
	23	tcp	filtered	telnet
	80	tcp	open	http
	139	tcp	filtered	netbios-ssn
	445	tcp	filtered	microsoft-ds
	2323	tcp	filtered	3d-nfsd
	4662	tcp	filtered	edonkey
	6346	tcp	filtered	gnutella
	6699	tcp	filtered	napster
	9929	tcp	open	nping-echo
	31337	tcp	open	Elite

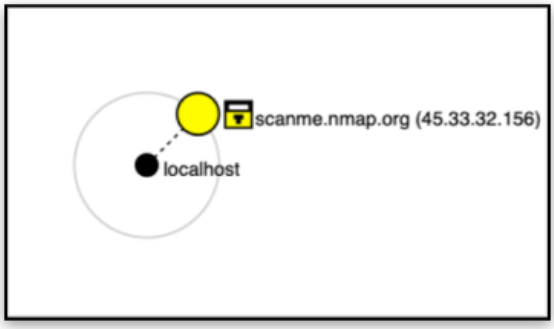

Figura 4-13: Herramienta *Zenmap* para el escaneo de puertos

4.5.3.1 Adaptación de los ataques al formato Weka para poder ser clasificados

Al igual que se ha realizado en la fase de diseño, va a ser necesario adaptar el tráfico malicioso obtenido a un formato que Weka pueda procesar. En este caso, la tarea será aún más complicada ya que no se dispone un texto plano con todos los flujos, como en el caso de los conjuntos de datos de la UPC. En este caso el objetivo es convertir un archivo de *Wireshark* (archivo *pcap*) con el tráfico malicioso capturado, a un conjunto de flujos con los mismos parámetros que los conjuntos de datos con el tercer tipo de filtrado (es el que se ha visto en el apartado anterior que da mejores resultados).

Para conseguir este objetivo se ha desarrollado un script en *Python* que mediante la herramienta *Scapy*, una herramienta de manipulación de paquetes soportada por *Python*, se obtendrán cada uno de los paquetes de tráfico del archivo *pcap* directamente y se organizará por flujos. Estos flujos contendrán paquetes en los que las IP de origen y destino coincidían, así como los puertos de origen y destino. Simplemente en el script se va recorriendo cada uno de los paquetes y comprobando sus direcciones *IP* y sus puertos. Después se dispondrá de una lista para cada flujo con los parámetros que necesitamos obtener ya vistos en las anteriores fases (puertos, tiempo inicio y final, flags etc…). Esta lista se irá actualizando siempre que se encuentren paquetes pertenecientes a un mismo flujo o se creará un nuevo flujo con su propia lista de parámetros (si sus direcciones *IP* y puertos no coinciden con ninguno de los que se han recorrido hasta el momento).

Finalmente, para cada uno de los paquetes, se imprimen los parámetros que interesan, todos dispuestos de forma continua, siguiendo así el mismo formato que tienen los conjuntos de datos de la UPC. Así solo se necesitaría introducir directamente el tráfico malicioso dentro de los conjuntos de datos para así probar a poder clasificar entre tráfico malicioso y tráfico normal.

4.5.3.2 Pruebas de clasificación con distintos data sets y filtrados de campos necesarios

Una vez el tráfico malicioso ha sido adaptado a un formato que Weka puede procesar (al igual que se realizó con los conjuntos de datos cedidos por la UPC), se van a realizar diversas pruebas para encontrar un clasificador para este tráfico y que sea lo más eficiente posible.

Primero, se tendrá que añadir un parámetro adicional tanto a los conjuntos de datos de la UPC como al trafico donde se ha producido el ataque. En el caso de los conjuntos de la UPC este parámetro siempre tendrá el valor de *normal* (al ser conjuntos tan grandes, simplemente se añadirá este nuevo parámetro haciendo uso de *AWK*). Y en el caso del tráfico malware, este parámetro se añadirá en el script de extracción de flujos anteriormente mencionado y tomará el valor de *malware-<tipo de ataque>*. De esta forma se generarán árboles que no sólo detecten si el tráfico ha sufrido un ataque o no, si no qué tipo de ataque en concreto.

Para la primera de las simulaciones realizadas se seguirá el mismo esquema que se ha observado que proporciona mejores resultados. Se utilizará el primer conjunto de datos para la etapa de entrenamiento y el quinto conjunto para la etapa de test. De forma adicional se aplicará el tercer tipo de filtrado comentado durante este capítulo.
Dentro del primer conjunto se introducirán todos los tipos de tráfico malicioso (después de haber sido adaptados al formato ARFF con el script de extracción de flujos).
El resultado es el representado en la figura 4-14:

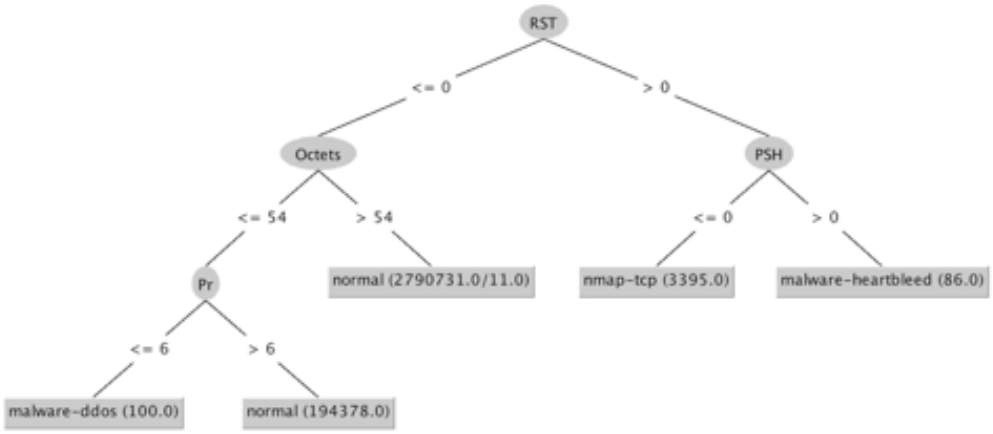

Figura 4-14: Primer árbol obtenido

Como se puede observar la detención de ataques como el tipo denegación de servicio es correcta. El clasificador se guía primero por el número de bytes (en este tipo de ataques se envían muchos paquetes de muy poco tamaño, así que los flujos tendrán pocos bytes), y finalmente depende de si pertenece al protocolo es TCP o no.
Comentar que en las etiquetas del árbol se nos proporciona información sobre las muestras clasificadas de forma correcta. (57/2) significaría que 57 muestras se han clasificado correctamente y 2 no.

Sin embargo, para poder realizar árboles de clasificación más fiables, se van a filtrar los flags TCP de los conjuntos para que así el programa no se guie por ellos.

Anteriormente se ha descrito en qué consiste el tipo de ataque de escaneo de puertos TCP (*nmap*). En este tipo de ataque, si el puerto del atacado no está activo, este envía un paquete con el flag RST activado. Es por esto por lo que el árbol se guía mucho por este flag, y es que, existen muchos puertos que no están activados por lo que el atacante recibe muchos RST, algo que el atacante no puede manipular para no ser detectado. Por ello, que el modelo de clasificación decida en primera instancia en base a este flag, es algo positivo.

El flag que se va a probar a filtrar es el *PSH* y el resultado es el represetado en la figura 4-15:

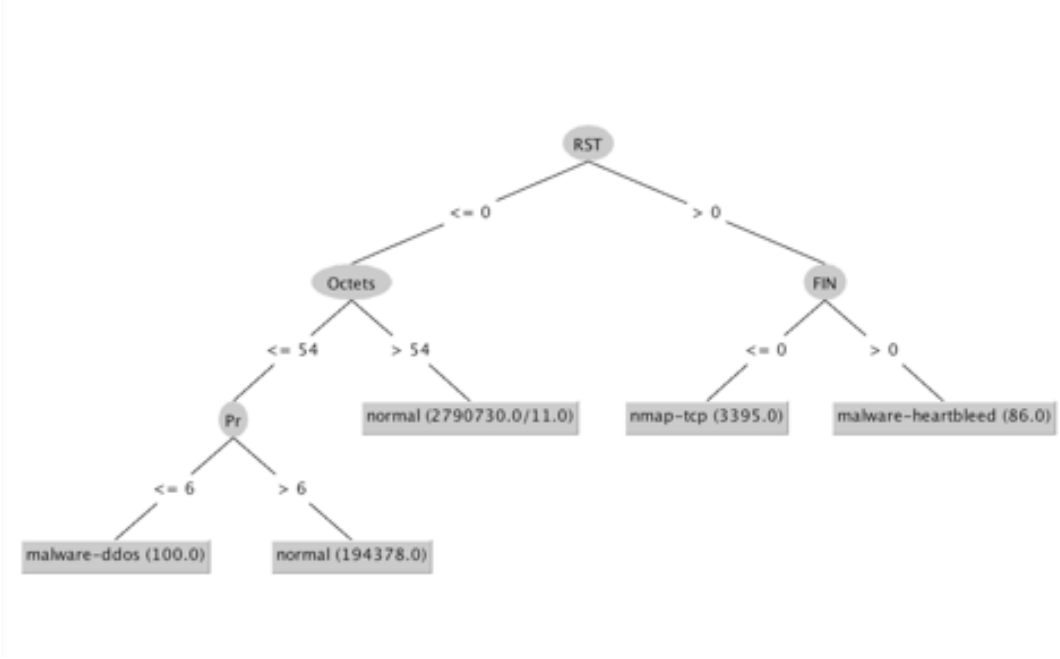

Figura 4-15: Árbol filtrando flag PSH

Como se puede observar, el árbol sigue decidiendo en base a un flag que el atacante puede manipular para no ser detectado. Así que a continuación se filtrará también este flag *FIN*.

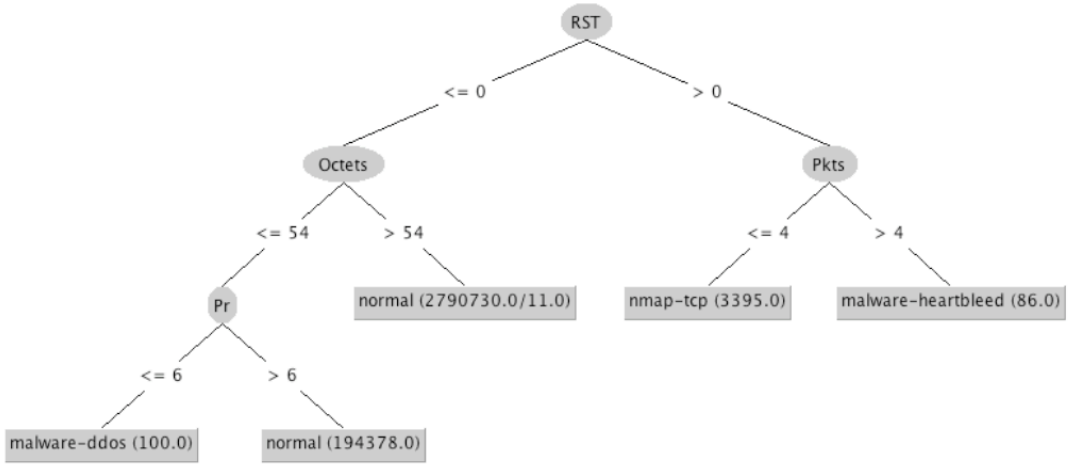

Figura 4-16: Árbol filtrando flag PSH y FIN

La clasificación ahora se realiza mediante el número de paquetes por flujo. Estos resultados son totalmente normales ya que, en el ataque *nmap-tcp,* el atacante envía un gran número de peticiones de un solo paquete preguntando por si están activos los puertos solicitados.

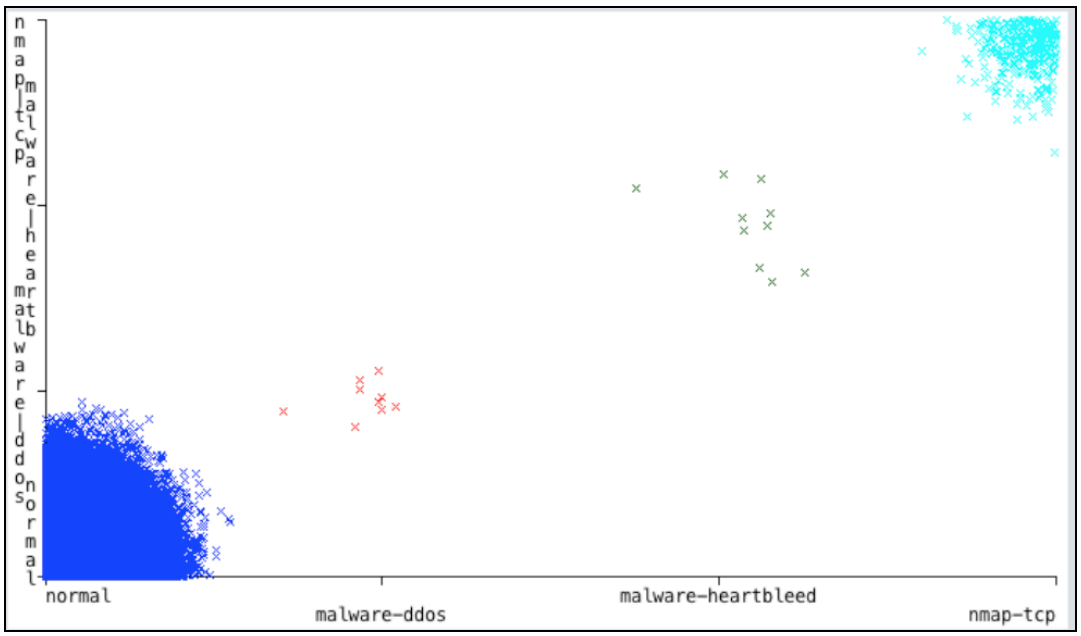

Figura 4-17: Resultados clasificación malware

Durante todas estas simulaciones, en el caso de los ataques de escaneo de puertos, sólo se ha utilizado uno de ellos.

Ahora se realizará una prueba de clasificación más realista. Permanecerán en el primer conjunto de la UPC los ataques *ddos*, *heartbleed* y *nmap_tcp*. En el quinto conjunto, conjunto que se usará para el testeo permanecerán el ataque *nmap_tcp* al dominio *hackthissite.org* pero se etiquetará como el ataque *nmap_tcp* a la página *scanme.nmap.org*, es decir, ambos con la misma etiqueta *nmap_tcp* aun siendo ataques distintos, pero del mismo tipo.

El árbol resultante es el mismo que hemos visto en el último caso. Pero se obtiene una etapa de test mucho más real (se comprobará cómo se comporta el calificador para un ataque distinto, pero del mismo tipo el cual se ha usado para la construcción del árbol de clasificación. Esto se asemeja más a como se comportaría el árbol de clasificación en la realidad.

```
=== Summary ===

Correctly Classified Instances     7148576              100      %
Incorrectly Classified Instances         3                0      %
Kappa statistic                          0.9993
Mean absolute error                      0
Root mean squared error                  0.0005
Relative absolute error                  0.2611 %
Root relative squared error              3.6535 %
Total Number of Instances          7148579

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 1,000    0,001    1,000      1,000    1,000      0,999    0,999     1,000     normal
                 ?        0,000    ?          ?        ?          ?        ?         ?         malware-ddos
                 ?        0,000    ?          ?        ?          ?        ?         ?         malware-heartbleed
                 0,999    0,000    1,000      0,999    0,999      0,999    0,999     0,999     nmap-tcp
Weighted Avg.    1,000    0,001    1,000      1,000    1,000      0,999    0,999     1,000

=== Confusion Matrix ===

       a       b       c       d   <-- classified as
 7146336       0       0       0 |       a = normal
       0       0       0       0 |       b = malware-ddos
       0       0       0       0 |       c = malware-heartbleed
       3       0       0    2240 |       d = nmap-tcp
```

Figura 4-18: Resultados de la efectividad del clasificador generado

Como se puede observar, la clasificación es casi perfecta, con solo tres muestras clasificadas de forma incorrecta del *nmap-tcp* realizado a la página hackthissite.org.

Recordar que el árbol se construye con un ataque *nmap-tcp* distinto al que se usa para testear por lo que se puede comprobar que, una vez construido el árbol para un tipo de ataque, funciona para más ataques de ese tipo.

4.5.3.3 Desarrollo de un script para poder aplicar los resultados de los árboles de clasificación

Una vez se ha obtenido un árbol de clasificación que permite identificar distintos tipos de ataques, se va a probar, de forma simplificada, la eficacia de este clasificador. Para ello se ha elaborado un script que, simplemente dado un flujo, sigue los caminos definidos por el árbol de clasificación (simples condiciones de cada parámetro dentro de los flujos) y decide si el tráfico ha sufrido un ataque o no.

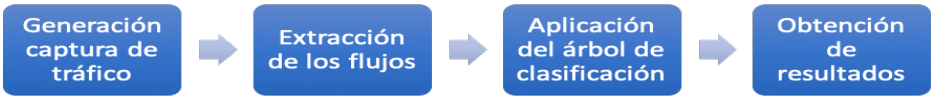

Figura 4-19: Cadena para la aplicación de los árboles de clasificación generados

Este script devuelve, en un archivo de texto externo, si se ha producido un ataque o no y en qué flujo se ha producido. Comentar que, de momento, solo se ha aplicado este script para ataques de tipo de escaneo de puertos. Es decir, solamente está implementado la parte que

ataques *nmap* del árbol de clasificación. Esta parte solo está hecha para demostrar, de forma simple, que se pueden aplicar los resultados de un árbol de clasificación. Para el resto de ataques, o si se desea añadir aún más tipos de ataques u otro tipo de flujos para detectar, se haría de la misma forma.

La cadena para llegar a detectar este tipo de ataques sería la siguiente: primero se captura el tráfico con un programa para este tipo de tareas como es *Wireshark*, para así generar un archivo de tipo *pcap*. De este archivo extraeremos los flujos con el script comentado en la anterior fase. Una vez hecho esto y teniendo ya el árbol de clasificación de ataques generado, aplicaremos este árbol a los flujos que tenemos (mediante el script que acabamos de describir). Por último, obtendríamos los resultados de esta clasificación.

Para finalizar, comentar que se ha probado este script, capturando tráfico del campus de la UAM, y además realizando un ataque de escaneo de puertos desde otro terminal. Como resultado se obtiene que, gracias a las condiciones creadas por el árbol de clasificación, se puede detectar este tipo de ataques.

4.6 Conclusiones

Se han obtenido, tras la realización de diversas simulaciones, unos resultados lo suficientemente satisfactorios como para poder afirmar que hemos construido un modelo correcto y fiable para la clasificación de flujos de tráfico en internet. Se ha determinado la combinación correcta entre conjunto de entrenamiento y conjunto de test para así, obtener los mejores resultados. También, se ha observado que la dependencia entre muestras juega un papel fundamental a la hora de la construcción de un árbol de clasificación.

Después, al haber desarrollado el mejor tipo de filtrado y haber encontrado la mejor combinación de los conjuntos de datos usados, se han analizado los resultados obtenidos.
A partir de estos resultados se ha determinado cómo, aquellas aplicaciones y servicios que poseen un puerto fijo y que se encuentran de forma abundante, tanto en el conjunto de entrenamiento como en el conjunto de test, tiene una mayor precisión a la hora de ser clasificadas. Sin embargo, existen flujos con una muy baja precisión. Esto puede deberse a factores como son la poca abundancia de este tipo de tráfico en los conjuntos de datos recogidos o que simplemente este tipo de aplicaciones están diseñadas específicamente para ser prácticamente indetectable.

En cuanto a la aplicación concreta de los resultados obtenidos, la clasificación de ataques, se ha podido comprobar que resulta muy útil tener ya, para los conjuntos de datos que tenemos, el clasificador que proporciona los resultados más óptimos. Esto ha ahorrado el hacer muchos más tipos de simulaciones al incluir el tráfico con ataques.

También, se ha podido observar que el programa Weka, para clasificar este tipo de ataques, se guía por los parámetros más obvios. El problema es que estos parámetros, aunque sería la forma más sencilla de diferenciar distintos tipos de ataques, puede ser que puedan ser evitados o manipulados por los atacantes, o simplemente no es lógico clasificar con ellos. Para solucionar este problema, se han tenido que realizar diversos tipos de filtrados (partiendo del tercer tipo de filtrado que se ha visto en la fase anterior que proporciona los mejores resultados) para así obtener el árbol de clasificación más óptimo.

Por último, se ha podido observar que, teniendo un árbol de clasificación ya creado, es muy sencillo aplicar estos resultados a una situación real y realizar diferentes pruebas de su

funcionamiento. Esto demuestra la sencillez de las técnicas de aprendizaje automático a la hora de clasificar tráfico y flujos, no solo aplicaciones, si no también ataques en la red u otro tipo de usos concretos que se le podrían dar a esta tecnología que sin duda es de gran importancia dentro de este campo de estudio.

5 Conclusiones y trabajo futuro

5.1 Conclusiones

A modo de resultado final, se han obtenido una serie de conclusiones muy satisfactorias, derivadas de la realización de este trabajo:

- **Eficacia del uso de técnicas de aprendizaje automático**: Mediante este tipo de técnicas evitamos otras más obsoletas como lo pueden ser el escaneo de puertos u otras técnicas que pueden tener riesgos relacionados con la privacidad ya que se basan en la inspección de la información que contienen los paquetes, información que puede y debería ser privada.
  En todos los flujos usados, tan sólo se usa información que no representa ningún riesgo para la privacidad, simples medidas estadísticas, suficientes para poder elaborar un árbol de clasificación eficiente.

- **Adaptación de los conjuntos de datos y parametrización del software Weka:**
  Se ha estudiado el formato *ARFF* para poder adaptar los conjuntos de datos al formato Weka mediante el uso de un script que analiza todos los flujos que se encuentran en los conjuntos y así listarlos dentro de la cabecera del archivo ARFF que tomará el software Weka.
  Para la elaboración de estos árboles de clasificación se ha usado el software Weka, el cual ha sido necesario entender su funcionamiento para así poder programar y trabajar con él, según las necesidades requeridas, variando todos los tipos de simulaciones que ofrece para la creación de árboles de clasificación y aprovechando su herramienta gráfica para, así, poder realizar los diferentes tipos de filtrados que se han descrito.

- **Resultados del desarrollo de árboles de clasificación:** Como conclusión se ha observado que somos capaces de generar un árbol de clasificación lo suficientemente fiable (como se puede observar en la sección de resultados sobre el desarrollo de árboles de clasificación de aplicaciones) como para poder ser usado en un caso real.
  Se ha estudiado, en el proceso, el papel que juega la dependencia de datos en la eficiencia de un clasificador y qué campos resultan más relevantes. Se ha observado con que conjuntos de datos obtenemos los mejores resultados y que con tipo de filtrados de los conjuntos de datos obtendríamos estos resultados.
  Por último, de forma adicional, se ha comprobado que aplicaciones que quedan definidas por un parámetro fijo como es el número de puerto y que poseen una gran cantidad de datos dentro de los conjuntos, tendrán una mayor precisión a la hora de ser clasificados. Los clasificadores se construirán hacia este tipo de aplicaciones, lo que hace que sean más fácilmente detectables.

- **Aplicación concreta de los resultados de la clasificación, la detención de malware:** Una vez obtenidos los resultados de los árboles de clasificación y qué combinaciones de conjuntos de datos, bajo qué condiciones, dan los mejores resultados, se ha estudiado una aplicación concreta, la clasificación de malware.
  Para ello se ha generado tráfico malicioso de forma artificial simulada y con diferentes herramientas desarrolladas se ha adaptado para que el software Weka

pueda procesarlo (extrayendo los diferentes flujos dentro del tráfico malicioso) y se ha mezclado con los conjuntos de datos usados en la anterior fase para que se pueda generar un árbol de clasificación lo más fiel a la realidad posible.

Una vez se adaptó el tráfico, durante todo este trabajo se han hecho varias pruebas y simulaciones (entendiendo cómo funcionan los ataques artificiales realizados) para, así, encontrar el clasificador más óptimo.

En un principio, hemos podido observar como el programa Weka se guía, para la clasificación, por parámetros muy sencillos o que pueden ser muy fáciles de evitar por un atacante. Pero realizando varios filtrados de campos, se ha podido elaborar un árbol de clasificación fiable y que se guía por parámetros inevitables por un atacante y que, por lo tanto, siempre sería detectado.

Para finalizar, a modo de inicio para un trabajo futuro, se ha desarrollado un script para poder aplicar los resultados de los árboles de clasificación generados. De esta forma y con una cadena de pasos de aplicar varios de los scripts desarrollados en este trabajo, se podría capturar tráfico real, obtener los flujos de datos y comprobar, con el árbol de clasificación ya creado y entrenado de forma correcta, si está sucediendo un ataque en el tráfico capturado. Algo sobre lo que se han realizado pruebas para comprobar su eficacia y se ha comprobado cómo funciona correctamente.

5.2 Trabajo futuro

Como se ha podido comprobar, existen multitud de parámetros posibles a la hora de justificar el por qué un clasificador va a ser mejor que otro.
A modo de trabajo futuro, sería de gran utilidad estudiar más acerca de que parámetros resultan útiles para la clasificación aplicada a otros conjuntos de datos de tráfico totalmente distinto (En el caso de este trabajo, se ha podido investigar acerca de estos parámetros, pero con conjuntos de datos de la UPC, tráfico recogido del campus de la universidad).

También sería muy interesante contar con más tipos de ataques y probar su comportamiento al crear árboles de clasificación con estos nuevos tipos de ataques. Así se podría observar que parámetros son más relevantes a la hora de clasificar este tipo de ataques.

Se podría desarrollar más el último script creado que aplica el árbol de clasificación ya creado, ya que, como se ha comentado, este script solo ha sido creado a modo de prueba para los ataques de escaneo de puertos. Sin embargo, no solo se podría aplicar los resultados obtenidos para la detención de ataques, si no que, una vez se tiene el árbol de clasificación resultante, se pueden utilizar los resultados para clasificar cualquier tipo de flujos (*malware*, flujos *dns*, *ftp*, etc.)

Para finalizar, comentar que la aplicación de los resultados obtenidos en la fase de creación de árboles de clasificación para la detención de malware, es sólo un caso concreto. Existen multitud de aplicaciones posibles, en el ámbito de análisis de tráfico, a las que aplicar las técnicas de aprendizaje automático para así mejorar las técnicas actuales hacia técnicas más fiables.

Referencias

[1] Tomasz Bujlow, Valentín Carela-Español and Pere Barlet-Ros: "Independent Comparison of Popular DPI Tools for Traffic Classification", Computer Networks 76 (2015), pp. 75-89. ©Elsevier.

[2] Valentín Carela-Español, Tomasz Bujlow, and Pere Barlet-Ros: "Is Our Ground-Truth for Traffic Classification Reliable?", In Proceedings of 15th International Passive and Active Measurement Conference (PAM 14), Los Angeles, CA, USA, March 2014 and Lecture Notes in Computer Science, vol. 8362, 2014, pp. 98-108. ©Springer

[3] Valentín Carela-Español, Pere Barlet-Ros, Albert Cabellos-Aparicio, and Josep Solé-Pareta: "Analysis of the impact of sampling on NetFlow traffic classification", Computer Networks 55 (2011), pp. 1083-1099. ©Elsevier.

[4] García, García, Juan Luis. Septiembre 2018. *"Concurrencia de flujos según el mix de aplicaciones de red"* (Trabajo fin de máster). Universidad Autónoma de Madrid.

[5] Department of Computer Science, University of Waikato. *"Attribute-Relation File Format (ARFF)"*. 2008. https://www.cs.waikato.ac.nz/~ml/weka/arff.html

[6] Eibe Frank, Mark A. Hall, and Ian H. Witten (2016). The WEKA Workbench. Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques", Morgan Kaufmann, Fourth Edition, 2016.

[7] EliteDataScienc. Septiembre 2017. *"Overfitting in Machine Learning: What it is and how to prevent it"*. https://elitedatascience.com/overfitting-in-machine-learning. Consultado en octubre de 2018.

[8] Sierra Araujo, B. (2006). Aprendizaje automático: Conceptos básicos y avanzados. Aspectos básicos utilizando el software Weka. Madrid: Pearson Prentice Hall.

[9] Yúbal FM, Genbeta.com. 2016. *"Así es Freenet, deep web alternative a Tor e I2P"*. https://www.genbeta.com/a-fondo/asi-es-freenet-deep-web-alternativa-a-tor-e-i2p. Consultado en abril de 2019.

[10] TechTarget. 2017. *"Deep Packet Inspection"*. https://searchnetworking.techtarget.com/definition/deep-packet-inspection-DPI. Consultado en julio de 2018.

[11] Kaspersky. *"¿Qué son los ataques DDoS?"*. https://latam.kaspersky.com/resource-center/threats/ddos-attacks. Consultado en enero de 2019

[12] Valentín Carela-Español, Pere Barlet-Ros, and Josep Solé-Pareta (Septiembre 2014)*: "Network Traffic Classification: From Theory to Practice"*. Universitat Politécnica de Catalunya BarcelonaTech.

[13] Ian H. Witten. "Data Mining with Weka". Department of Computer Science, University of Waikato, New Zealand. weka.waikato.ac.nz

[14] https://nmap.org/, consultado en febrero de 2019.

[15] Herramienta *Zenmap* para el escaneo de puertos. https://nmap.org/zenmap/, consultado en febrero de 2019.

[16] Request for Comments: 3514. S. Bellovin, AT&T Labs Research. "The Security Flag in the IPv4 Header. 1 April 2003.

Anexos

A Funcionamiento del programa Weka

El funcionamiento del programa es sencillo de entender y la cantidad de opciones, tanto de clasificación como de modificación de los conjuntos de datos sin tener que interactuar de forma directa con ellos, lo hace un programa de suma utilidad para este trabajo.
Si se ejecuta el programa y a su vez se ejecuta la aplicación de explorador se accede a una versión más gráfica que del programa.

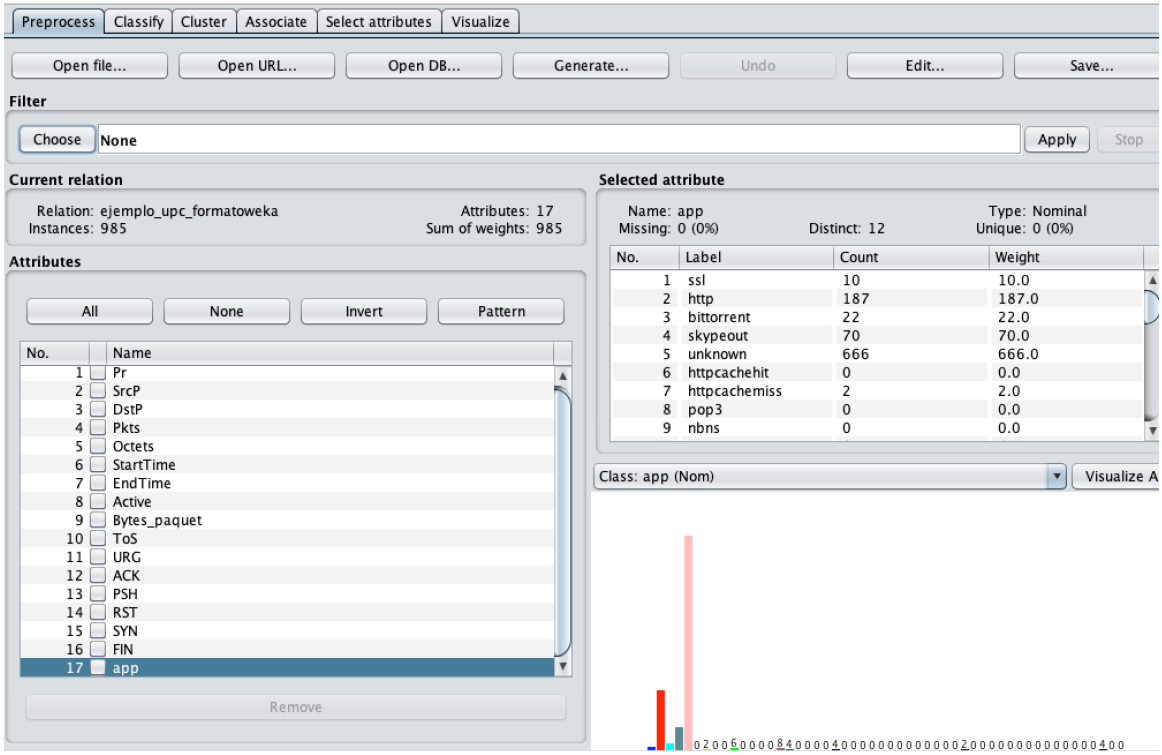

Figura 0-1: Ventana principal del programa

En esta primera ventana es donde se carga el conjunto de datos. Como se puede observar en la figura, el programa nos proporciona información acerca del número atributos (número de parámetros extraídos de cada flujo) así como el número de instancias (número de flujos dentro de un conjunto, en este caso son 985 ya que se trata de un conjunto de datos de prueba).
Como se puede observar también, sin filtrar ninguno de los parámetros, existen un total de 17 los cuales ya han sido descritos anteriormente.

De forma adicional el programa también enumera todos los atributos en la parte izquierda, desde la cual seleccionar cada uno de ellos y eliminarlos totalmente. Esto provoca que en el conjunto de datos se elimine ese atributo de cada flujo (algo de gran utilidad, ya que, no siempre es beneficioso tener todos los atributos y eliminarlos de forma manual sería una tarea imposible de realizar).

La herramienta de poder eliminar cada uno de los atributos es usada con grann frecuencia y que mejorará en gran medida los clasificadores construidos.

Esta ventana, como vemos en su parte derecha, también incluye una lista de los valores que toman cada parámetro. En el ejemplo de la figura podemos ver algunas de las aplicaciones que se encuentran pertenecientes al parámetro "app" dentro de este conjunto de datos.

A continuación, existe la ventana de clasificación, una vez realizadas todas las modificaciones y filtraciones sobre el conjunto de datos sobre el que vamos a trabajar.

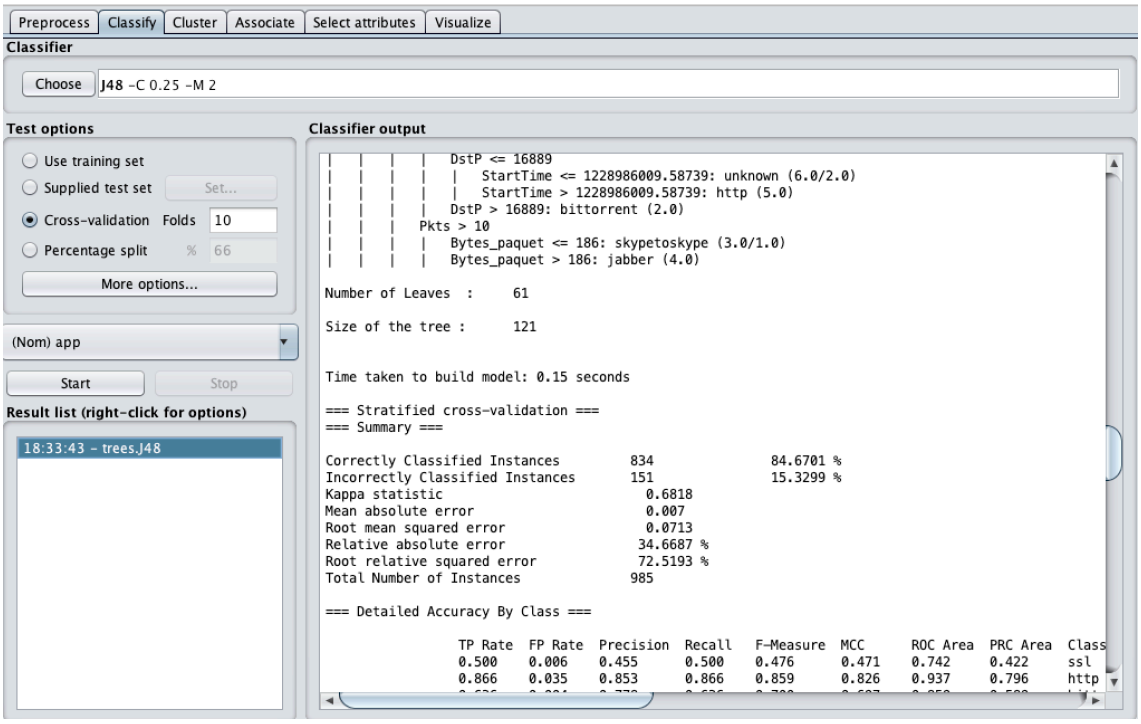

Figura 0-2: Ventana de clasificación del programa

Existen multitud de opciones en esta ventana. En primer lugar, en la parte superior izquierda se puede elegir el tipo y algoritmo de clasificador. Durante todas las pruebas, se ha escogido un tipo de árbol de clasificación llamado J48.

Existen una serie de opciones muy importantes en la parte de *Test Options*. En ellas se puede elegir el modo de realizar la clasificación y como funcionarán tanto la etapa de entrenamiento como de test.

B Tipos de simulaciones del programa Weka

Dentro del software Weka existen cuatro (aunque como se observará realmente se trata de tres opciones) opciones distintas de tipos de simulación. En ellas se puede elegir el modo de realizar la clasificación y como funcionarán tanto la etapa de entrenamiento como de test. Y es que durante la construcción del clasificador se realizan dos fases. La primera de ellas, la fase de entrenamiento, en la cual a partir del conjunto de datos se construye un árbol de clasificación. La segunda de estas fases es la fase de test que, como su nombre indica, testea el árbol creado en la anterior etapa y proporciona amplios resultados sobre este testeo.

La primera de estas opciones mencionadas permite utilizar un conjunto de datos para la fase de entrenamiento, esta opción será irrelevante durante este trabajo ya que, abriendo el conjunto de datos en la ventana anterior ya conseguimos el mismo resultado. La segunda opción permite utilizar otro conjunto de datos distinto al utilizado para construir el árbol de clasificación (es necesario que contengan los dos el mismo número de atributos) para realizar la fase de testeo.

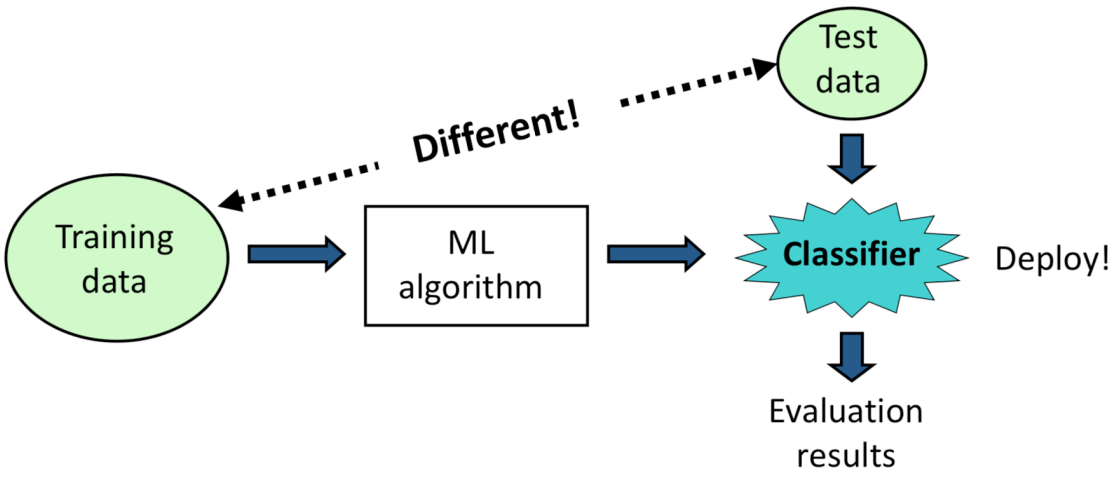

Figura 0-3: Utilización de conjuntos de datos diferentes para la clasificación [13]

La tercera opción permite, dado un 10% del conjunto de datos para test, ir variando este 10% dentro del mismo conjunto para de esta forma aumentar sustancialmente la precisión. El número de iteraciones es un parámetro libre de modificación.

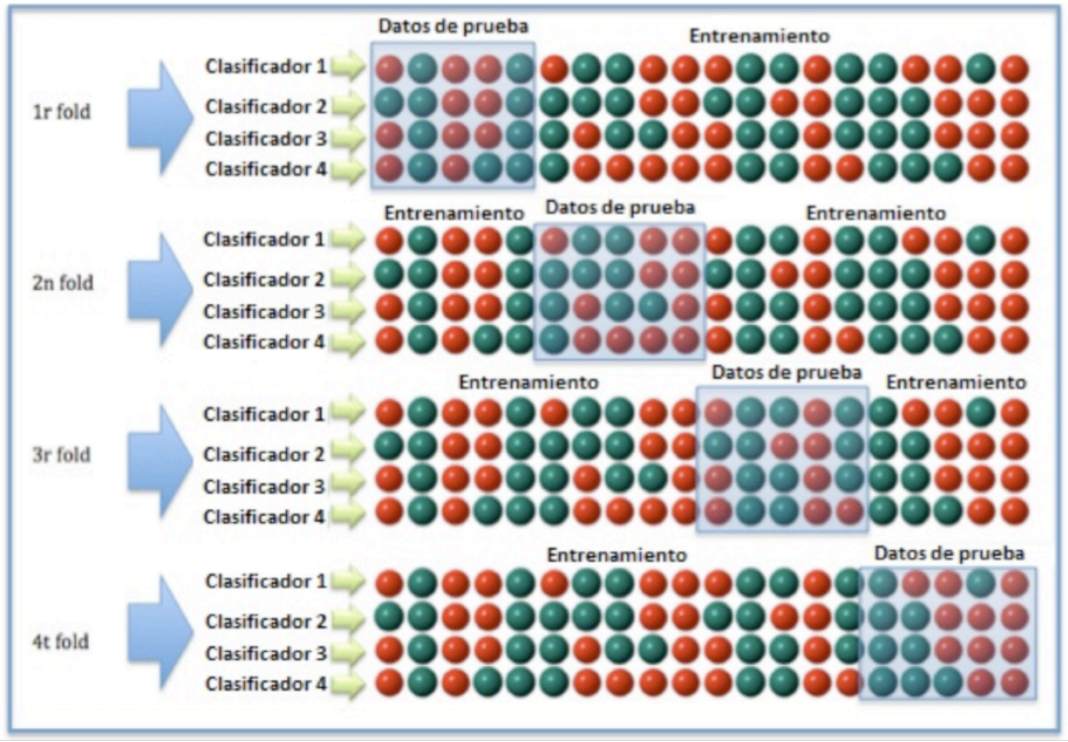

Figura 0-4: Validación cruzada [13]

La cuarta de estas opciones, permite dividir el conjunto de datos seleccionado en la ventana anterior en una parte del mismo para realizar la etapa de entrenamiento y la otra parte para la etapa de test. El porcentaje asignado para cada parte se puede modificar manualmente con total libertad.

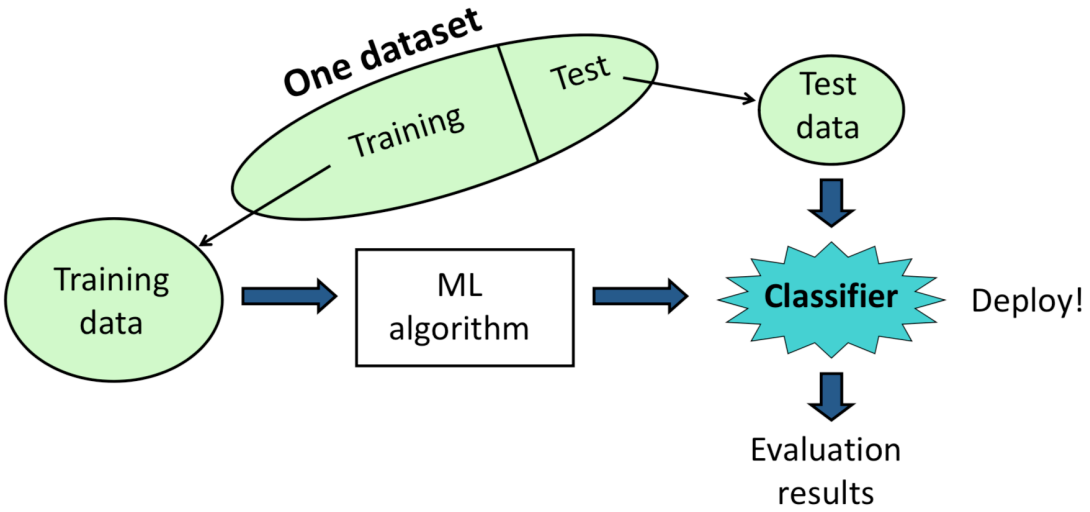

Figura 0-5: División de un conjunto de datos para la etapa de entrenamiento y la de test [13]

C Resultados obtenidos en las distintas simulaciones

```
Number of Leaves  :     44946

Size of the tree :      89891


Time taken to build model: 3513.15 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 10.62 seconds

=== Summary ===

Correctly Classified Instances      273052               91.4716 %
Incorrectly Classified Instances     25458                8.5284 %
Kappa statistic                          0.8865
Mean absolute error                      0.0049
Root mean squared error                  0.054
Relative absolute error                 16.0905 %
Root relative squared error             43.8871 %
Total Number of Instances           298510
```

Figura 0-6: División del conjunto UPC-1 en un 90% para la etapa de entrenamiento y un 10% para la etapa de test

```
Number of Leaves  :     45834

Size of the tree :      91667


Time taken to build model: 3575.43 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 1.26 seconds

=== Summary ===

Correctly Classified Instances       27369               91.6854 %
Incorrectly Classified Instances      2482                8.3146 %
Kappa statistic                          0.8894
Mean absolute error                      0.0048
Root mean squared error                  0.0533
Relative absolute error                 15.8283 %
Root relative squared error             43.2474 %
Total Number of Instances           29851
```

Figura 0-7: División del conjunto UPC-1 en un 99% para la etapa de entrenamiento y un 1% para la etapa de test

```
Time taken to build model: 3733.73 seconds

=== Evaluation on test set ===

Time taken to test model on supplied test set: 102.74 seconds

=== Summary ===

Correctly Classified Instances     2869613               85.1744 %
Incorrectly Classified Instances    499488               14.8256 %
Kappa statistic                          0.7961
Mean absolute error                      0.007
Root mean squared error                  0.0724
Relative absolute error                 23.8129 %
Root relative squared error             60.3793 %
Total Number of Instances          3369101
Ignored Class Unknown Instances                 4
```

Figura 0-8: UPC-1 para la etapa de entrenamiento y UPC-2 para la etapa de test

```
=== Evaluation on test set ===

Time taken to test model on supplied test set: 57.1 seconds

=== Summary ===

Correctly Classified Instances     1088643               81.1551 %
Incorrectly Classified Instances    252792               18.8449 %
Kappa statistic                          0.7513
Mean absolute error                      0.0086
Root mean squared error                  0.0819
Relative absolute error                 28.8301 %
Root relative squared error             67.4039 %
Total Number of Instances          1341435
Ignored Class Unknown Instances                 2
```

Figura 0-9: UPC-1 para la etapa de entrenamiento y UPC-3 para la etapa de test

```
=== Evaluation on test set ===

Time taken to test model on supplied test set: 98.61 seconds

=== Summary ===

Correctly Classified Instances     2592309               85.8351 %
Incorrectly Classified Instances    427793               14.1649 %
Kappa statistic                          0.8171
Mean absolute error                      0.0069
Root mean squared error                  0.0699
Relative absolute error                 22.2861 %
Root relative squared error             55.8663 %
Total Number of Instances          3020102
Ignored Class Unknown Instances                12
```

Figura 0-10: UPC-1 para la etapa de entrenamiento y UPC-4 para la etapa de test

```
=== Evaluation on test set ===

Time taken to test model on supplied test set: 247.2 seconds

=== Summary ===

Correctly Classified Instances     6166767               86.3946 %
Incorrectly Classified Instances    971139               13.6054 %
Kappa statistic                          0.8289
Mean absolute error                      0.0067
Root mean squared error                  0.0691
Relative absolute error                 21.1062 %
Root relative squared error             53.9242 %
Total Number of Instances          7137906
Ignored Class Unknown Instances              8430
```

Figura 0-11: UPC-1 para la etapa de entrenamiento y UPC-5 para la etapa de test

```
Time taken to build model: 4189.87 seconds

=== Evaluation on test set ===

Time taken to test model on supplied test set: 28.91 seconds

=== Summary ===

Correctly Classified Instances      538217               80.814
Incorrectly Classified Instances    127778               19.186
Kappa statistic                          0.7609
Mean absolute error                      0.0088
Root mean squared error                  0.0815
Relative absolute error                 27.6689 %
Root relative squared error             63.1608 %
Total Number of Instances           665995
Ignored Class Unknown Instances                 5
```

Figura 0-12: UPC-1 para la etapa de entrenamiento y UPC-7 para la etapa de test

D Descripción de los ataques usados para la creación de árboles de clasificación de flujos de tráfico malicioso.

En este anexo, se describirán los dos ataques usados que han sido extraídos de forma libre en internet. Es necesario comprender su funcionamiento para poder entender los árboles de clasificación resultantes en las diferentes pruebas realizadas. Sin embargo, este tipo de ataques suelen ser comunes y su funcionamiento es ya conocido.

El resto de ataques, al haber sido generados personalmente con diferentes herramientas para la elaboración de este trabajo, sí que quedan estudiados fuera del anexo, en su capítulo correspondiente.

En primer lugar, se dispone de un ataque de tipo denegación de servicio [11]. Este tipo de ataque es muy conocido y muy común, pero se ha concluido que es interesante incluirlo para que el árbol de clasificación construya un modelo en base a este tipo de tráfico y poder clasificarlo. Comentar, de forma breve, de que se trata un ataque de denegación de servicio. Este tipo de ataques provoca la saturación de puertos de uno o varios servidores mediante el envío de múltiples flujos de información (de pocos paquetes) en un espacio muy reducido de tiempo.

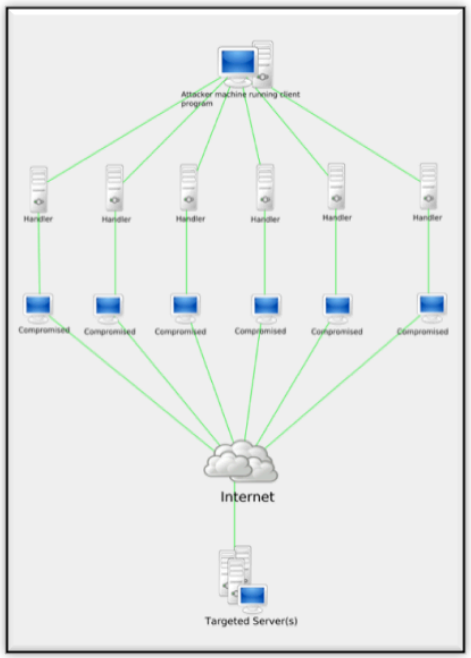

Figura 0-13: Esquema ataque DDOS [11]

Este ataque es obtenido de forma libre de internet al igual que el siguiente que se describirá.

El siguiente ataque se trata del tipo *Heartbleed*. Este tipo se trata de un fallo en la seguridad de la biblioteca de código abierto *OpenSSL*. Permite a un atacante leer la memoria de un servidor o cliente, permitiendo así el acceso a información confidencial. Este ataque también ha sido obtenido de forma libre en internet.